Universidad Politécnica de Cartagena

industriales
etsii UPCT

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE ILUMINACIÓN BASADO EN DISPOSITIVOS MÓVILES INTELIGENTES Y MÓDULOS EMPOTRADOS DE BAJO COSTE.

Titulación: Ingeniero Industrial
Alumno/a: Adrián Colmena Mateos
Director/a/s: Juan Antonio López Riquelme
Andrés Iborra García

Cartagena, 19 de abril de 2017

Agradecimientos

Quiero dar mi más sincero agradecimiento a Juan Antonio López Riquelme, por haberme guiado en este proyecto de final de carrera. Por todas las atenciones, por el tiempo que ha invertido conmigo, y sobre todo por su apoyo y paciencia. Sin él nada de esto hubiera sido posible.

También me gustaría destacar la importancia en el proyecto de Andrés Colmena, por haberme guiado en el aprendizaje de las nuevas áreas del conocimiento que he enfrentado en este proyecto, por su calma a responder todas mis preguntas y por su gran ayuda en la instalación del prototipo.

Por último, no me quiero olvidar de mis compañeros de facultad, familia y amigos, que gracias a ellos todo es siempre mucho más fácil.

Gracias a todos.

Índice

Capítulo 1

Introducción

1.1 Introducción.

Durante la pasada década la comunicación entre seres humanos ha dado un gran paso gracias a los avances tecnológicos. Actualmente, en una sociedad avanzada e inteligente, la innovación consiste en la comunicación del usuario, ya no sólo con otros usuarios, sino con el medio que le rodea.

La posibilidad de un entorno inteligente, que proporcione respuesta al usuario mediante la conexión a los distintos dispositivos que le rodean de manera automática es algo que cada vez se hace más necesario. Permitir que la persona sea un elemento más de un sistema donde los elementos se envían información de una manera rápida, segura y eficiente, es quizás el mayor avance que pueda estar teniendo la tecnología en estos momentos.

El gran número de personas que disponen de al menos un dispositivo móvil inteligente, tipo *Smartphone*, abre un campo de posibilidades infinito de como este dispositivo puede recibir información del medio, procesarla, mostrársela al usuario e incluso interactuar de forma automática con elementos del entorno para hacer la vida del usuario, no sólo más cómoda y segura para él, sino incluso pensar en poder cuidar el propio ambiente que le rodea.

Por otro lado, hasta ahora los sistemas *GPS* nos permitían localizarnos y orientarnos en grandes espacios, pero su precisión era reducida y perdía toda capacidad en espacios interiores y reducidos. Pero gracias a la tecnología *iBeacon,* es posible ya poder situar a un

usuario dentro, por ejemplo, de un centro comercial y permitir que su *Smartphone,* reaccione según la zona por donde el sujeto se esté moviendo.

Debido a esta creciente tendencia, existen actualmente diversas plataformas que hacen posible el desarrollo de aplicaciones propias para conectar a los usuarios mediante esta tecnología, y dar una nueva visión del marketing. No sólo en el ámbito de la publicidad, sino en el ámbito de la automatización de sistemas la mayor comodidad del trabajador en la empresa, además del ahorro de recursos que puede aportar esta automatización.

1.2 Objetivos.

El objetivo principal de este proyecto es el diseño de un sistema automático de iluminación mediante el uso de dispositivos de bajo coste, que permitan localizar al usuario dentro de espacios reducidos. Para ello, se proponen los siguientes sub-objetivos:

- Estudio, tanto a nivel hardware como software, de los posibles elementos para diseñar el sistema.
- Diseño de un actuador inalámbrico de bajo coste para regular la iluminación.
- Diseño de un sistema de localización de usuarios de bajo coste para interiores.
- Diseño de una App para dispositivos móviles inteligentes, que permita localizar al usuario e interactuar con el sistema completo.
- Desplegar y validar el funcionamiento del sistema en condiciones reales.
- Redacción del documento final de memoria de proyecto.

1.3 Desarrollo de la memoria.

Capítulo 1: Introducción

El Capítulo 1 presenta la motivación de este trabajo, además de enumerar los objetivos del mismo y describir la memoria presentada.

Capítulo 2. Estado del Arte

En el Capítulo 2 se expone el panorama actual de las nuevas tecnologías de sistemas inteligentes, se describen los dispositivos empotrados de bajo coste más importantes existentes en el marcado, algunos de los microordenadores más utilizados en el mercado, los

dispositivos móviles inteligentes más populares y los protocolos de comunicación más innovadores y usados en estos sistemas.

Capítulo 3. Descripción del sistema.

En este capítulo se seleccionan los dispositivos empotrados, el microordenador y dispositivo móvil elegido. Posteriormente, se describen todos los elementos y protocolos utilizados con un mayor nivel de detalle.

Capítulo 4. Caso de estudio. Hardware y software desarrollado.

Se desarrolla un caso de estudio concreto en el que se emplean los elementos seleccionados en el capítulo anterior. Se detallan las características principales del software y el hardware desarrollado en este proyecto.

Capítulo 5. Conclusiones y trabajos futuros.

En este capítulo se enumeran las principales conclusiones tras la realización de este trabajo, así como los futuros trabajos a desarrollar.

Capítulo 2

Estado del Arte

2.1 Introducción

Un *Sistema Inteligente* es aquel que permite recoger información de lo que ocurre en el entorno y reaccionar ante esos estímulos. En concreto, un sistema inteligente completo incluye "sentidos" que le permiten recibir información de su entorno, puede actuar, y tiene una memoria para archivar el resultado de sus acciones. Tiene un objetivo e, inspeccionando su memoria, puede aprender de su experiencia. Aprende cómo lograr mejorar su rendimiento y eficiencia.

Todo sistema inteligente tiene un dispositivo maestro, un HUB o concentrador, que actúa como controlador de las señales y del resto de dispositivos empotrados que haya alrededor. Para implementar este dispositivo se puede usar un ordenador de placa reducida o SBC (*Single Board Computer*).

Por otro lado, para el control de las luces de manera directa necesitaremos algún tipo de dispositivo empotrado que reciba la orden y encienda cada lámpara. Este tipo de dispositivo es común que esté basado en alguna placa que incluye un micro-controlador, y que además podría incluir un dispositivo de comunicación (WiFi, Bluetooth, etc.).

2.2 Dispositivos de localización

Para seleccionar los dispositivos mencionados anteriormente se tendrán en cuenta aspectos como el consumo, el tamaño, el precio y las prestaciones, entre otros.

En primer lugar, el sistema necesita de un conjunto de sensores para determinar la posición del usuario. Existen diferentes dispositivos en el mercado que permiten la localización de interiores, entre los que se destacan los siguientes:

2.2.1 RFID.

RFID (siglas de *Radio Frequency IDentification,* en español identificación por radiofrecuencia) es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transpondedores o tags RFID. El propósito fundamental de la tecnología RFID [RFID] es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las denominadas Auto ID (*automatic identification*, o identificación automática).

Las etiquetas RFID son unos dispositivos pequeños, similares a una pegatina, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID. Las etiquetas pasivas no necesitan alimentación eléctrica interna, mientras que las activas sí la requieren. Una de las ventajas del uso de radiofrecuencia (en lugar, por ejemplo, de infrarrojos) es que no se requiere visión directa entre emisor y receptor.

Ilustración 2.1.- ***Tarjeta RFID.***

2.2.2 iBeacon

iBeacon [iBeacon] es un sistema de posicionamiento en interiores (IPS *indoor positioning system*) que Apple Inc. denomina como "una nueva clase de transmisores de bajo consumo y bajo coste que pueden notificar a dispositivos iOS7 de su presencia por proximidad". También pueden ser empleados por el sistema operativo Android. Esta tecnología permite a los dispositivos iOS u otro hardware enviar notificaciones *push* a los dispositivos móviles inteligentes próximos.

iBeacon se basa en Bluetooth de baja energía (BLE), también conocido como Bluetooth *Smart*. Bluetooth LE se puede encontrar en dispositivos Bluetooth 4.0 que soporten modo dual. En un escenario de la vida real sería más un sensor inalámbrico de posición/contexto, que puede determinar con precisión su posición en una tienda: los iBeacons podrían enviar notificaciones de elementos alrededor del sensor que están en venta y que se podrían estar buscando, y podría permitir pagos en el punto de venta (POS) donde no se necesita sacar un monedero o tarjeta para efectuar el pago.

De alguna forma podría ser un competidor de la tecnología NFC, ya que emplea Bluetooth LE *proximity sensing* para transmitir un identificador único universal recogido por un aplicación compatible o sistema operativo, y que puede ser convertido en una localización física o generar una acción en el dispositivo como por ejemplo un check-in en una red social.

Las balizas o *beacons* pueden ser implementadas en diferentes formatos, incluidos los dispositivos alimentados por pilas de botón, así como USB sticks.

Siguiendo los pasos de sus anteriores sistemas, Estimote lleva ahora el uso de balizas electrónicas para móviles a un nuevo nivel con su nuevo sistema de localización de interiores. *Estimote* proporciona al usuario la posibilidad de crear sus propias aplicaciones, con plantillas existentes tanto para *iOS* como para *Android,* lo que representa una gran ventaja a tener cuenta para elegir a esta compañía y sus *beacons*.

Ilustración 2.2.- Vista de un beacon de Estimote.

2.2.3 NFC

Near field communication (NFC, comunicación de campo cercano en español) [NFC] es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos. Los estándares de NFC cubren protocolos de comunicación y formatos de intercambio de datos, y están basados en ISO 14443 (RFID, *radio-frequency identification*) y FeliCa.

Como en ISO 14443, NFC se comunica mediante inducción en un campo magnético, en donde dos antenas de espiral son colocadas dentro de sus respectivos campos cercanos. Trabaja en la banda de los 13,56 MHz, lo que hace que no se aplique ninguna restricción y no requiera ninguna licencia para su uso.

Soporta dos modos de funcionamiento, que deben proporcionar todos los dispositivos del estándar NFCIP-1:

- Activo: ambos dispositivos generan su propio campo electromagnético, que utilizarán para transmitir sus datos.

- Pasivo: sólo un dispositivo genera el campo electromagnético y el otro se aprovecha de la modulación de la carga para poder transferir los datos. El iniciador de la comunicación es el encargado de generar el campo electromagnético.

El protocolo NFCIP-1 puede funcionar a diversas velocidades como 106, 212, 424 ó 848 kb/s. Según el entorno en el que se trabaje, los dispositivos involucrados en la comunicación pueden configurar este parámetro, así como realizar un reajuste del mismo en cualquier instante del intercambio de información.

2.3 Control de iluminación.

A la misma vez que el sistema debe posicionar al usuario, por otra parte, una serie de dispositivos se encargarán del control de la iluminación. En concreto, son necesarios dos tipos de dispositivos: el encargado de actuar de forma individual sobre cada elemento de iluminación y el responsable de coordinar todos los anteriores, así como de permitir un intercambio de mensajes con otros dispositivos locales y remotos de forma inalámbrica.

2.3.1 Dispositivos empotrados.

Un sistema empotrado es cualquier dispositivo que contiene una computadora, pero no es una computadora de usos generales como las que normalmente conocemos. Comúnmente este aparato para procesamiento de información está incluido dentro de un

producto o sistema más grande. Es por esta razón que se llaman empotrados. Algunas de las funciones que se esperan de un sistema empotrado son que respondan, monitoricen o ayuden a controlar el ambiente que los rodea mediante mecanismos de entrada y salida, tales como sensores. Es por esto que son creados para relacionarse con sus alrededores.

Los sistemas empotrados son parte de lo que se conoce como "*Ubiquitous Computing*" o computación ubicua. Este término se refiere a poder acceder a información en cualquier momento y en cualquier lugar. Esto es una muestra de que vivimos en la sociedad de la información y los sistemas empotrados son parte de ese estilo de vida.

Algunas de las características de estos sistemas es que son dependientes. Dentro del concepto de dependencia surgen otros como confiabilidad, mantenimiento, seguridad y disposición. Todos ellos se pueden aplicar en sistemas empotrados que funcionan en fábricas o plantas industriales que están en funcionamiento durante todo el día. También, los sistemas empotrados tienen que ser eficientes, tanto en el consumo de energía como el código que ocupan para funcionar. Esto se debe a la limitada capacidad de memoria de almacenamiento que poseen. También se dice que es importante el tamaño y peso de estos dispositivos, ya que en la actualidad lo que se busca es que sean portátiles y de un fácil acceso a la población.

2.3.1.1 Arduino

Arduino [Arduino] (ver Ilustración 2.3) es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 y ATmega32u4. Todos ellos son sencillos y tienen un bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado, el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el *bootloader* que se ejecuta en la placa.

Desde octubre de 2012, Arduino también se usa con microcontroladores CortexM3 de ARM de 32 bits, que coexistirán con las más limitadas, pero también económicas placas AVR de 8 bits. ARM y AVR no son plataformas compatibles a nivel binario, pero se pueden programar con el mismo IDE de Arduino y hacer programas que compilen sin cambios en las dos plataformas. Eso sí, las microcontroladoras CortexM3 usan 3,3 V, a diferencia de la mayoría de las placas con AVR, que generalmente usan 5 V. Arduino se puede utilizar para desarrollar objetos interactivos autónomos, o puede ser conectado a software tal como Adobe Flash, Processing, Max/MSP, Pure Data, entre otros). Las placas se pueden

montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

Arduino puede tomar información del entorno a través de sus entradas y controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en *Wiring*) y el entorno de desarrollo Arduino (basado en *Processing*). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un computador.

Ilustración 2.3.- Vista del sistema Arduino UNO.

Arduino permite conectar diferentes extensiones o *shields* sobre la placa principal, aportando funcionalidades extra al dispositivo sin perder además las conexiones originales (salvo algunas excepciones). En la Ilustración 2.4 se muestra un ejemplo de conexión de la placa Arduino UNO con dos placas de accesorios o shields: Ethernet Shield y un shield para dotar a la placa de Arduino UNO de comunicación GSM/GPRS.

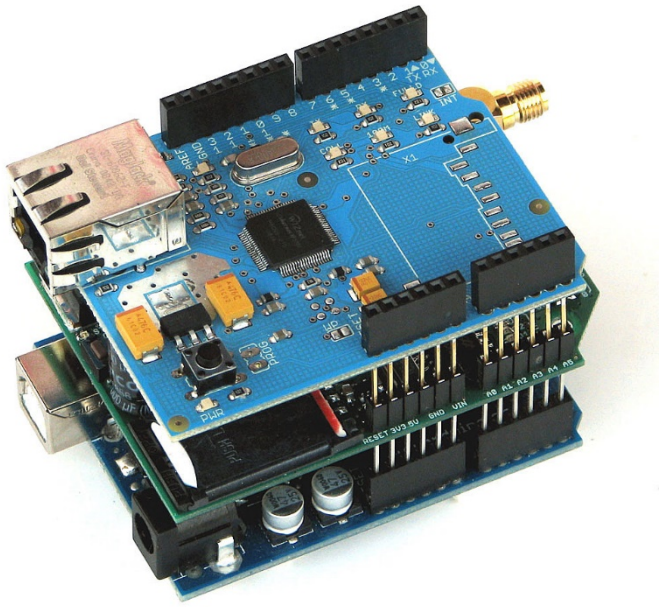


Ilustración 2.4.- Placa Arduino UNO con dos shields conectados.

2.3.1.2 Módulos WiFi ESP

Son chips WiFi de bajo coste que incorporan una pila TCP/IP completa, además de poder programarse para la aplicación o caso de estudio deseado.

Hay varias versiones de este chip, siendo la placa más conocida y utilizada la versión denominada ESP8266 [ESP8266].

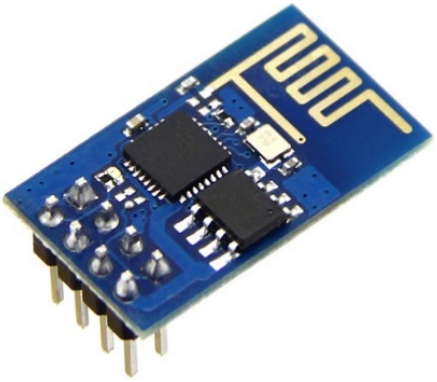


Ilustración 2.5.- Vista del módulo ESP8266.

Actualmente hay otra versión más moderna, que aún no está del todo instalada en el mercado que es el ESP32. Representa ciertas ventajas, como la incorporación de BLE, mayor memoria RAM y capacidad de procesamiento.

Ilustración 2.6.- Vista del módulo ESP32.

2.3.2 SBC's

Para ejercer el control en nuestro sistema se requiere de un dispositivo de cómputo que actúe como servidor, y también como controlador. Una buena solución sería el uso de un ordenador de placa reducida, o *SBC*. Una placa computadora u ordenador de placa reducida es una computadora completa en un único circuito. El diseño se centra en un solo microprocesador con la RAM, E/S y todas las demás características de un computador funcional en una sola tarjeta que suele ser de tamaño reducido, y que tiene todo lo que necesita en la placa base. Las más utilizadas son:

2.3.2.1 Raspberry Pi

Raspberry Pi [RPi] (ver Ilustración 2.7) es un ordenador de placa reducida o (placa única) de bajo coste, desarrollado en Reino Unido por la Fundación Raspberry Pi. El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (aunque existe la posibilidad de realizar *overclock* de hasta 1 GHz), un procesador gráfico (GPU) VideoCore IV y 512 MB de memoria RAM. El diseño no incluye un disco duro ni unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente. Tampoco incluye fuente de alimentación ni carcasa. En febrero de 2012 la fundación empezó a aceptar órdenes de compra del modelo A, en febrero de 2013 del modelo B y en julio de 2014 el modelo B+. La evolución de un modelo a otro ha sido la adición de puertos USB en cada versión (1 en el modelo A, 2 en el modelo B y 4 en el modelo B+), el incremento de la RAM (de 256 MB en el modelo A hasta 512 MB en el B) y la aparición del puerto Ethernet en la versión B. Además, en la última versión se ha reducido el consumo del sistema.

Dispone de conexiones de audio (Jack 3,5 mm), Ethernet, USB, microUSB para alimentación (5 V), ranura para tarjeta micro-SD, salida HDMI y hasta 40 pines GPIO (*General Purpose Input/Output*) en el último modelo.

En cuanto al software, la fundación da soporte para las descargas de las distribuciones para arquitectura ARM Raspbian (derivada de Debian), RISC OS 5, Arch Linux ARM (derivado de Arch Linux) y *Pidora* (derivado de *Fedora*), y promueve principalmente el aprendizaje del lenguaje de programación *Python* y otros lenguajes como *Tiny BASIC, C* y *Perl*. Sin embargo, es posible instalar muchos otros sistemas operativos e incluso se ha llegado a implementar Android.

La conexión a Internet se puede realizar mediante una conexión Ethernet, a través del puerto incluido en la placa (a partir del modelo B), o mediante WiFi, conectando un *dongle* USB WiFi.

Ilustración 2.7.- Vista *Raspberry* Pi 2 modelo B.

2.3.2.2 BeagleBone

BeagleBone [BeagleBone] (ver Ilustración 2.8) es un ordenador de placa reducida o credit-card-sized basada en el kernel 3.8 de Linux. Es un tipo de BeagleBoard creado por Texas Instruments con una filosofía open-source y de bajo coste. Apareció por primera vez en octubre de 2011 y posteriormente, en abril de 2013, apareció un modelo actualizado llamado BeagleBone Black.

Implementa un microprocesador ARM Cortex-A8 a 720 MHz ó 1 GHz (dependiendo del modelo), hasta 512 MB DDR3 de memoria RAM, acelerador gráfico 3D y memoria flash eMMC de 2 GB.

En cuanto a conectividad, dispone de conexión Ethernet, microHDMI, puerto USB, USB OTG (*On The Go*), ranura microSD, estéreo Jack (input y output), JITAG, Socket de alimentación (5 V), puerto para cámara, 2 buses CAN (*Controller Area Network*) y puertos de expansión.

BeagleBone es compatible con diferentes sistemas operativos. Los creadores de este sistema, han creado distribuciones de *Android* y *Linux*. Por otro lado, basados en Linux, están disponibles Angstrom Distribution, Ubuntu, Debian, ArchLinux, Gentoo, Sabayon, Buildroot, Erlang y Fedora. Otros disponibles son QNX y FreeBSD.

Al igual que otros sistemas como Arduino o Raspberry Pi, tiene dos posibilidades de conexión a Internet mediante cable Ethernet conectado en el puerto disponible en la placa y mediante WiFi, a partir de la conexión por USB de un dongle WiFi.

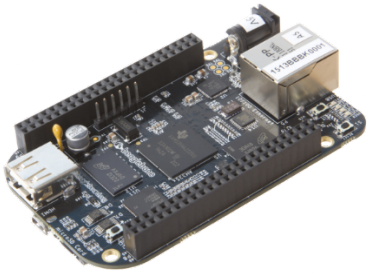


Ilustración 2.8.- *Vista BeagleBone Black.*

2.3.2.3 Intel Compute Stick

Intel Compute Stick [Intel Compute Stick] (ver Ilustración 2.9) es un diminuto dispositivo del tamaño de un paquete de chicles que puede transformar cualquier televisor o pantalla HDMI* en un completo ordenador. Disponible con una amplia gama de procesadores Intel, ofrece todo lo que te gusta de un equipo de sobremesa en un dispositivo que cabe en la palma de la mano.

Se trata de un miniPC completo que es compatible con sistemas operativos como Windows 8.1 o Ubuntu.

Ilustración 2.9.- ***Vista Intel Compute Stick.***

2.4 Dispositivos móviles inteligentes.

Un dispositivo móvil inteligente es un dispositivo electrónico, por lo general conectado a otros dispositivos o redes a través de diferentes protocolos como Bluetooth, NFC, Wi-Fi, 3G, X10, etc, que puede funcionar hasta cierto punto de forma interactiva y autónoma.

La opinión generalizada es que este tipo de dispositivos superarán a cualquier otra forma de computación inteligente y de comunicación, en un tiempo muy corto. Varios dispositivos notables a la hora de escribir son los teléfonos inteligentes como el iPhone de Apple o la mayoría de los dispositivos con sistema operativo Android, tablets y tabletas, como el iPad de Apple o el Google Nexus 7.

Los más populares actualmente son los Smartphones, donde hay una gran variedad de marcas, pero quizás lo que más define a un Smartphone es el sistema operativo que utilice.

Un sistema operativo móvil o SO móvil es un sistema operativo que controla un dispositivo móvil al igual que los PCs que utilizan Windows o Linux. Los dispositivos móviles tienen sus sistemas operativos como Android, iOS entre otros. Los sistemas operativos móviles son mucho más simples y están más orientados a la conectividad inalámbrica, los

formatos multimedia para móviles y las diferentes maneras de introducir información en ellos. Mencionar también que los sistemas operativos utilizados en los dispositivos móviles están basados en el modelo de capas.

2.4.1 Sistemas operativos para móvil

2.4.1.1 Android

El sistema operativo Android [Android] es uno de los sistemas operativos más usados, está basado en Linux, diseñado originalmente para cámaras fotográficas profesionales, luego fue vendido a Google y modificado para ser utilizado en dispositivos móviles como los teléfonos inteligentes y luego en tablets como es el caso del Galaxy Tab de Samsung. Actualmente se encuentra en desarrollo para usarse en netbooks y PCs. El desarrollador de este S.O. es Google, fue anunciado en el 2007 y liberado en el 2008; además de la creación de la Open Handset Alliance, compuesto por 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para celulares, esto le ha ayudado mucho a Google a masificar el S.O, hasta el punto de ser usado por empresas como HTC, LG, Samsung y Motorola, entre otros.

Android Inc. es la empresa que creó el sistema operativo móvil, se fundó en 2003 y fue comprada por Google en el 2005 y en 2007 fue lanzado al mercado. Su nombre se debe a su inventor, Andy Rubin.

Android está basado en Linux, disponiendo de un Kernel en este sistema y utilizando una máquina virtual sobre este Kernel, que es la responsable de convertir el código escrito en Java de las aplicaciones a código capaz de comprender el mismo.

Las aplicaciones para Android se escriben y desarrollan en Java, aunque con unas APIs propias, por lo que las aplicaciones escritas en Java para PC y demás plataformas ya existentes no son compatibles con este sistema. Es frecuente es uso de Android Studio como IDE de desarrollo para aplicaciones para este SO, aunque Android Studio es relativamente joven en este sector.

2.4.1.2 iOS

iOS [iOS] es el sistema operativo que da vida a dispositivos como el iPhone, el iPad y el iPod Touch. Su simplicidad y optimización son sus pilares para que millones de usuarios se decanten por iOS en lugar de escoger otras plataformas que necesitan un hardware más potente para mover con fluidez el sistema operativo. Cada año, Apple lanza una gran

actualización de iOS que suele traer características exclusivas para los dispositivos más punteros que estén a la venta en ese momento.

Anteriormente denominado iPhone OS y creado por Apple originalmente para el iPhone, posteriormente fue usado en el iPod Touch e iPad. Es un derivado de Mac OS X, se lanzó en el año 2007, aumento el interés con el iPod Touch e iPad, que son dispositivos con las capacidades multimedia del iPhone, pero sin la capacidad de hacer llamadas telefónicas. Su principal revolución es una combinación casi perfecta entre hardware y software, y el manejo de la pantalla multi-táctil, que no podía ser superada por la competencia hasta el lanzamiento del celular Galaxy S I y II por parte de Samsung.

Las aplicaciones para iOS se escriben y desarrollan en Swift usando el IDE de Xcode.

2.4.1.3 Windows Phone.

Anteriormente llamado Windows Mobile [Windows Phone] es un S.O. móvil compacto desarrollado por Microsoft, se basa en el núcleo del sistema operativo Windows CE y cuenta con un conjunto de aplicaciones básicas. Está diseñado para ser similar a las versiones de escritorio de Windows estéticamente y existe una gran oferta de software de terceros disponible para Windows Mobile, la cual se puede adquirir a través de la tienda en línea Windows Marketplace for Mobiles.

2.5 Protocolos de comunicación

Una vez establecidos todos los dispositivos del sistema, pasamos a la parte del enlace que se produzca entre ellos. El envío de información mediante los distintos elementos del entorno se producirá a través de las diferentes redes que se forman en el sistema. A continuación, se explicarán las redes y protocolos más usuales en la comunicación de dispositivos móviles inteligentes y dispositivos empotrados.

2.5.1 WiFi

WiFi [WiFi] (nombre común en español proveniente de la marca WiFi) es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Los dispositivos habilitados con WiFi (como una computadora personal, un televisor inteligente, una videoconsola, un teléfono inteligente o un reproductor de música) pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Dicho punto de acceso tiene un alcance de unos veinte metros en interiores, distancia que es mayor en el denominado espacio libre.

2.5.1.1 Características de WiFi

El estándar 802.11, en el que se basa la tecnología WiFi ha ido experimentando modificaciones con el paso de los años, ofreciendo nuevas características que se adecuan a las nuevas necesidades de los dispositivos que incorporan la tecnología WiFi. Las principales versiones de este protocolo son:

- IEEE 802.11b: trabaja en la banda de los 2,4 GHz y alcanza una velocidad de 11 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen, dando opción a que convivan distintas redes WiFi en el mismo espacio.

- IEEE 802.11g: trabaja en la banda de los 2,4 GHz y alcanza una velocidad de 54 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen.

- IEEE 802.11n: trabaja en la banda de los 2,4 GHz y opcionalmente en la de 5 GHz, alcanza una velocidad de entre 54 Mbps y 600 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen.

- IEEE 802.11ac: trabaja en la banda de los 5 GHz, alcanza una velocidad de 1300 Mbps, aunque al trabajar en una frecuencia mayor, la señal pierde un poco de alcance. El espectro electromagnético de este protocolo se divide en 23 canales que no se superponen.

Aunque el espectro electromagnético de WiFi se divide en varios canales, las distintas redes WiFi presentes en la misma zona pueden compartirlos sin interferirse entre ellas, ya que, aunque por los canales haya más paquetes de datos, como cada red tiene su propio SSID (*Service Set Identifier*), el cual consiste en 32 bytes que identifican una determinada red. De esta manera, los dispositivos sólo tienen que ignorar los paquetes que no vayan dirigidos al SSID de la red a la que pertenecen.

En cuanto al alcance de la señal, un punto de acceso que cumpla con los estándares 802.11b y 802.11g puede alcanzar unos 100 metros de alcance, por lo que WiFi se convierte en un buen sustituto de las comunicaciones cableadas en áreas locales.

La seguridad de los datos transmitidos inalámbricamente corre a cargo de los estándares de encriptación de datos WEP, WPA y WPA2. WPA utiliza el protocolo TKIP, el cual refuerza la seguridad proporcionada por WEP, aunque se ha comprobado que este protocolo también presenta vulnerabilidades, por lo que finalmente se recomienda utilizar WPA2, el cual utiliza AES (*Advanced Encryption Standard*) para proteger la red.

Por sus características, WiFi está presente en casi cualquier producto del mercado que necesite comunicarse inalámbricamente con otro dispositivo electrónico. Es el caso de los ordenadores de sobremesa, los portátiles, los smartphones, las tabletas, las Smart TV, las neveras, las bombillas inteligentes y todo tipo de productos que necesitan acceder a una red local para transmitir datos inalámbricamente. Es compatible con prácticamente todos los sistemas operativos actuales, tanto a nivel de escritorio, como son Windows, Mac OS X y Linux, como a nivel de dispositivos móviles como pueden ser Android, iOS y Windows Phone.

2.5.2 Bluetooth

La tecnología Bluetooth [Bluetooth] es un estándar que permite intercambiar datos de forma inalámbrica entre dos dispositivos situados a poca distancia entre sí. Es una tecnología muy expandida actualmente.

2.5.2.1 Características de Bluetooth

Bluetooth trabaja en una frecuencia situada entre los 2400 y los 2483,5 MHz de la banda ISM. Emplea una técnica de modulación en espectro ensanchado en la que la señal se emite sobre una serie de radiofrecuencias, sobre las cuales se va saltando de forma sincronizada con el transmisor. Esto evita que un receptor no autorizado sea capaz de escuchar la transmisión, ya que recibirá una señal ininteligible.

A lo largo de su historia, el estándar Bluetooth ha ido pasando por diferentes versiones, las cuales han ido dotando, generalmente, de mayor velocidad de transmisión y de nuevas características a los dispositivos que lo incorporaban. Las principales versiones son:

- Bluetooth 1.2: velocidad de transmisión de 1 Mbps.
- Bluetooth 2.0 + EDR: velocidad de transmisión de 1Mbps. Se introdujo el EDR (*Enhanced Data Rate*) para transmitir datos más rápidamente.
- Bluetooth 3.0 + HS: velocidad de transmisión de 24 Mbps y mejoras en la seguridad.
- Bluetooth 4.0: velocidad de transmisión de 24 Mbps. Introducción del bajo consumo.
- Bluetooth 4.1: es una modificación a nivel de software enfocada a mejorar el bajo consumo introducido en la especificación 4.0, por lo que no es necesario actualizar el hardware de los dispositivos que cumplan este estándar.

- Bluetooth 4.2: introduce características para interaccionar con los dispositivos del IoT (*Internet of Things*).

Bluetooth permite establecer medidas de seguridad a la hora de realizar una conexión, requiriendo la introducción de una clave de seguridad para que los dos dispositivos se queden emparejados correctamente.

2.5.2.2 Bluetooth Low Energy

Es una tecnología perteneciente a la especificación 4.0 de Bluetooth, enfocada en el bajo consumo. Por ello, está pensada para utilizarse en dispositivos que ejecuten tareas muy concretas de recolección de datos, pudiendo entrar en un modo de muy bajo consumo entre cada conexión.

Debido a esto, con Bluetooth de baja energía se pierden ciertas posibilidades que se tenían con la especificación clásica, como transmitir audio o archivos de un tamaño considerable. No obstante, la ventaja obtenida con el aumento de la autonomía de la batería, introduce a los dispositivos que incorporen esta tecnología en un nuevo segmento repleto de posibilidades de sensorización que antes era imposible que existiese.

- **Retrocompatibilidad**

  Al ser una especificación que introdujo un nuevo paradigma en el modo en el que la tecnología Bluetooth realizaba las comunicaciones, no es compatible con las anteriores versiones de Bluetooth. En concreto, Bluetooth SIG ha definido una serie de distinciones en función del tipo de protocolo que emplean los dispositivos para saber los dispositivos con los que son compatibles.

2.5.3 Comunicaciones cliente/servidor

2.5.3.1 Protocolo HTTP

Hypertext Transfer Protocol o HTTP [HTTP] (en español protocolo de transferencia de hipertexto) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que culminó en 1999 con la publicación de una serie de RFC, el más importante de ellos es el RFC 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener el estado. Para esto se usan

las cookies, que es la información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de "sesión", y también permite rastrear usuarios, ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

Es un protocolo orientado a transacciones y sigue el esquema de petición-respuesta entre un cliente y un servidor. El cliente (se le suele llamar "agente de usuario", en inglés *user agent*) realiza una petición enviando un mensaje con cierto formato al servidor. El servidor (se le suele llamar un servidor web) le envía un mensaje de respuesta. Ejemplos de cliente son los navegadores web y los spider.

- **Mensajes**

Los mensajes HTTP son en texto plano, lo que lo hace más legible y fácil de depurar. Esto tiene el inconveniente de hacer los mensajes más largos.

- Los mensajes tienen la siguiente estructura:

· Línea inicial (termina con retorno de carro y un salto de línea).

· Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.

· Para respuestas: La versión del HTTP usado seguido del código de respuesta (que indica que ha pasado con la petición seguido de la URL del recurso) y de la frase asociada a dicho retorno.

· Las cabeceras del mensaje que terminan con una línea en blanco. Son metadatos. Estas cabeceras le dan gran flexibilidad al protocolo.

· Cuerpo del mensaje. Es opcional. Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor Por ejemplo para una petición podría contener ciertos datos que se quieren enviar al servidor para que los procese. Para una respuesta podría incluir los datos que el cliente ha solicitado.

2.5.3.2 Protocolo MQTT

MQTT (*Message Queue Telemetry Transport*) [MQTT], un protocolo usado para la comunicación **machine-to-machine** (M2M) en el "**Internet of Things**". Este protocolo está orientado a la comunicación de sensores, debido a que consume muy poco ancho de banda y puede ser utilizado en la mayoría de los dispositivos empotrados con pocos recursos (CPU, RAM, entre otros). Un ejemplo de uso de este protocolo es la aplicación de Facebook Messenger tanto para Android como para iOS.

La arquitectura de MQTT sigue una **topología de estrella**, con un nodo central que hace de servidor o "broker" con una capacidad de hasta 10000 clientes. El broker es el encargado de gestionar la red y de transmitir los mensajes, para mantener activo el canal. Los clientes mandan periódicamente un paquete (**PINGREQ**) y esperan la respuesta del broker (**PINGRESP**). La comunicación puede ser cifrada entre otras muchas opciones.

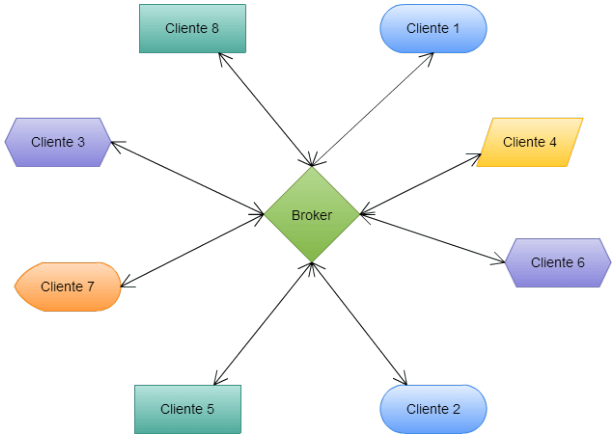


Ilustración 2.10.- ***Vista Esquema MQTT.***

La comunicación se basa en unos "**topics**" (temas), que el cliente que publica el mensaje crea y a los nodos que deseen recibirlo deben subscribirse. La comunicación puede ser de **uno a uno**, o de **uno a muchos**.

Un "topic" se representa mediante una cadena y tiene una estructura jerárquica. Cada jerarquía se separa con '/'. Por ejemplo, "edificio1/planta5/sala1/raspberry2/temperatura" o "/edificio3/planta0/sala3/arduino4/ruido". De esta forma se pueden crear jerarquías de clientes que publican y reciben datos, como podemos ver en la siguiente imagen:

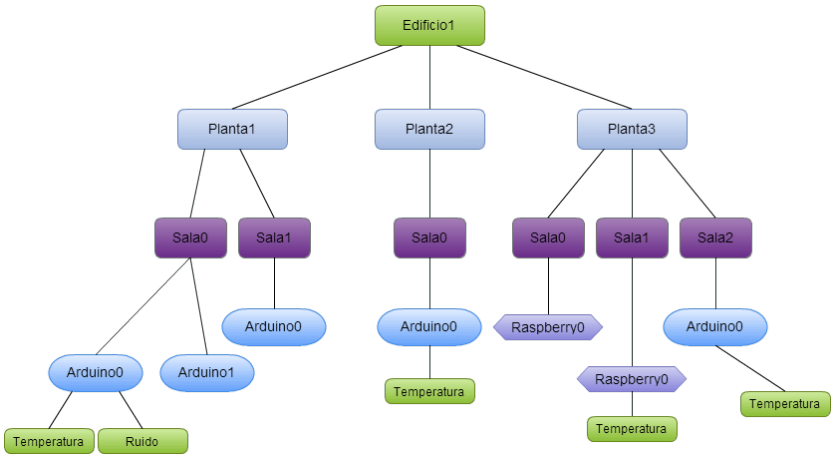


Ilustración 2.11.- ***Vista Ejemplo MQTT.***

De esta forma un nodo puede subscribirse a un "topic" concreto ("edificio1/planta2/sala0/arduino0/temperatura") o a varios ("edificio1/planta2/#").

Capítulo 3

Descripción del sistema

3.1 Introducción

En el capítulo anterior se han estudiado algunos dispositivos empotrados, microcontroladores y sistemas operativos para dispositivos móviles más importantes.

Dentro de las alternativas propuestas, se ha seleccionado la tarjeta *Raspberry Pi* como SBC, los módulos *ESP8266* como hardware de control para la iluminación e *iOS* como sistema operativo en el que se desarrollará la aplicación que conecte con los *Beacons*. También entre toda la oferta de *Beacons* que hay, se ha elegido el modelo ofrecido por *Estimote*.

Se ha decidido utilizar *Estimote* debido a la existencia de su software libre y por la sencillez de su configuración. En cuanto al sistema operativo para el *Smartphone* se ha elegido iOS por su mayor simplicidad a la hora de diseñar y desarrollar la App. Finalmente, la elección del módulo ESP8266 se debe a su simplicidad, bajo coste y ser suficiente para controlar una luz debido a las diversas líneas de E/S digitales que incopora.

En cuanto a los dispositivos empotrados, se han seleccionado Raspberry Pi 2 modelo B. Raspberry Pi, se trata de una opción de gran interés y es por ello que ha sido elegida para formar parte de este proyecto. Se trata de un dispositivo muy económico, de uso muy extendido y open-source. Esto hace que haya mucha información, proyectos y librerías por la web. Está además basada en Linux, por lo que es posible compilar programas de dicho sistema operativo. Todo esto sumado a que se pueden encontrar elementos que extienden

su funcionalidad (cámaras, baterías, alimentación mediante paneles solares, entre otros) hace de Raspberry Pi un sistema ideal para incorporar en Sistemas Inteligentes.

3.2 Arquitectura del sistema

La arquitectura de este proyecto va desde la conexión del iBeacon con el usuario hasta el encendido de la luz, pasando por varias fases.

Los puntos que se van a ir desarrollando son los diferentes elementos que se resumen a continuación:

- La baliza o *Beacon,* que servirá para detectar la posición de las personas que entren en la sala.
- La aplicación de *Smartphone,* en este caso para iOS, que permite la localización de los usuarios y el aviso de la zona en la que se encuentran.
- Un servidor o *bróker,* que servirá para la comunicación entre el móvil y los módulos WiFi y está implementado en la Raspberry Pi 2 (RBPi 2).
- Los componentes hardware para el control de la iluminación a través de los módulos ESP8266.

La arquitectura del sistema se basa primeramente en la detección y localización del usuario que entra en la zona mediante módulos *beacon*.

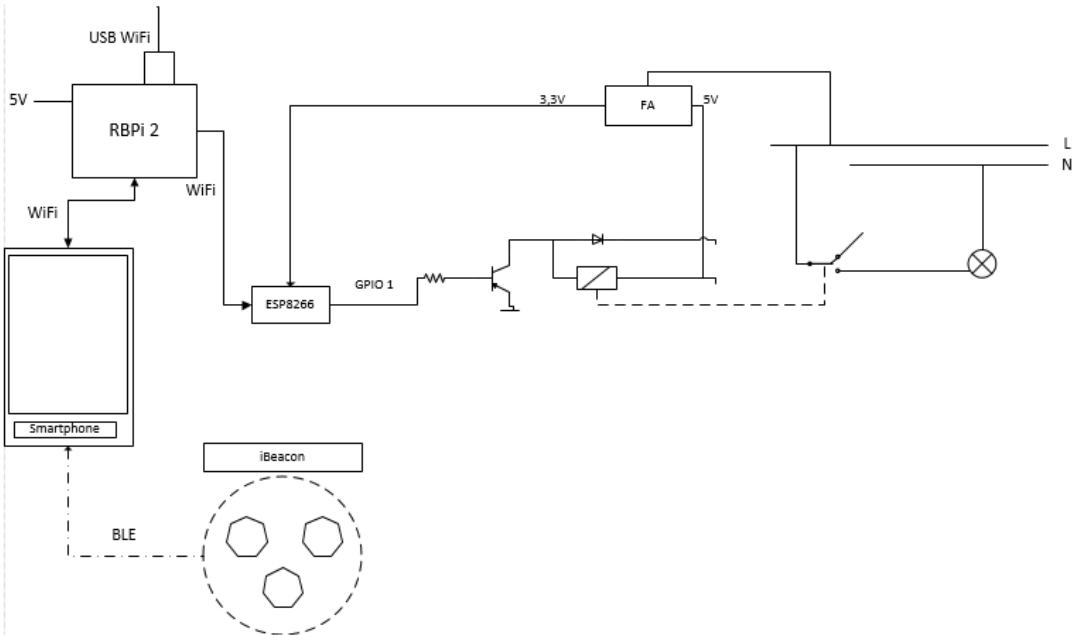


Ilustración 3.1.- Esquema del sistema.

3.3 Beacons

Un **beacon** es un dispositivo de bajo consumo que emite una señal broadcast, y son suficientemente pequeños para fijarse en una pared o mostradores. Utiliza conexión Bluetooth de bajo consumo (BLE) para transmitir mensajes o avisos directamente a un dispositivo móvil sin necesidad de una sincronización de los aparatos. Concretamente, la señal es captada por estos dispositivos y se transmite a menudo a un servidor en la nube a través de Internet. El servidor de la nube procesa la información y lleva a cabo análisis más detallado para guiar los comportamientos basados en la localización específica del dispositivo móvil.

A diferencia de la tecnología GPS, los beacons pueden ser utilizados para la localización exacta dentro de un entorno cerrado. Existen numerosas aplicaciones que han surgido, como el marketing, siendo uno de los usos más mayoritarios. Otras como el servicio a clientes basadas en la localización, asistencia personalizada, etc.

Su implementación está presente en los sistemas operativos móviles más recientes. Android y iOS ya incorporan esta funcionalidad, gracias al soporte de BLE.

Ilustración 3.2.- Composición de un Beacon.

3.3.1 Hardware

El hardware consiste en un microcontrolador con un chip de radio Bluetooth LE y una batería, generalmente del denominado tipo botón. Los nuevos chips están optimizados para trabajar con BLE.

El chip de radio BLE es generalmente fabricado por dos grandes empresas: Texas Instruments y Nordic Semiconductor.

Generalmente las empresas proveedoras de beacons utilizan el hardware fabricado por los anteriores fabricantes mencionados, pero con sus propios firmware.

Las baterías de botón son las opciones más populares para la mayoría de estos dispositivos. Concretamente, se suelen emplear las de tecnología de iones de litio y con capacidades comprendidas entre 240 y 1000 mAh. También se pueden encontrar soluciones de beacons que están alimentados mediante baterías alcalinas tipo AA.

También se pueden encontrar soluciones comerciales que se pueden conectar en una toma de corriente o un puerto USB. Estos dispositivos no necesitan un reemplazo de baterías, con el inconveniente de la disponibilidad de una toma de corriente cercana.

3.3.2 Firmware

Cada beacon tiene un firmware específico, según el proveedor, que permite al hardware del beacon funcionar adecuadamente. El firmware puede controlar varias características que afectan a la batería.

Potencia de Transmisión (Tx Power)

Los beacons transmiten una señal con una potencia fija, conocida como Tx Power. A medida que la señal viaja en el aire la intensidad de la señal va disminuyendo con la distancia. Con un Tx Power superior se consiguen mayores alcances, lo que significa mayor consumo. Del mismo modo, si se configura un Tx Power menor se traduce a menor rango de alcance, pero menos consumo de batería.

Advertising Interval

La frecuencia con la que un beacon emite una señal se conoce como advertising interval. Un intervalo de 100 ms significa que la señal se emite cada 100 milisegundos, es decir, 10 veces por segundo. Un mayor intervalo, por ejemplo 500 ms, significa que la señal emite sólo dos veces por segundo, lo que significa menos consumo de energía. Cuando advertising interval aumenta la duración de la batería aumenta pero la capacidad de respuesta del dispositivo receptor disminuye. No hay una elección óptima del advertising interval, y las aplicaciones que necesiten baja latencia deben elegir intervalos más bajos, en cambio los que necesiten mayor duración de la batería necesitan un intervalo mayor.

En las especificaciones de Apple para iBeacon especifica que el advertising interval recomendado es de 100 ms.

Cada beacon ofrece su propia forma de configurar el hardware y los parámetros asociados. Algunos proveedores de beacons proporcionan su propia aplicación para el Smartphone para configurar los beacons. Otros beacons proporcionan una interfaz abierta a través de cualquier cliente GATT. La principal ventaja de utilizar GATT es que cientos de beacons pueden ser configurados a la vez.

3.3.3 Protocolos de beacons

Bluetooth Low Energy (BLE) tiene la capacidad de intercambiar datos en dos estados: modo conectado y modo advertising. En el modo Conectado se utiliza el atributo genérico (GATT) para transferir datos en una conexión. En el modo advertising se utiliza el perfil de acceso genérico (GAP) para transmitir datos a cualquiera que esté escuchando. Concretamente, el modo Advertising es una transferencia de uno a muchos y no tiene garantías sobre la coherencia de los datos. El BLE de los beacons aprovecha el modo Advertising GAP para transmitir datos en paquetes Advertising periódicos, especialmente formateados. Cada tipo de beacon utiliza una especificación personalizada para particionar los datos Advertising, dándoles un significado.

3.3.3.1 iBeacon

Es un protocolo creado por Apple, que se introdujo por primera vez en la Worldwide Developers Conference 2013, y que opera sobre la tecnología (BLE) fue creada por Nokia.

iBeacon utiliza BLE para transmitir un identificador único universal (UUID), que es recogido por una aplicación o sistema operativo compatible con el protocolo. El identificador más otros bytes enviados se pueden usar para identificar la posición física del dispositivo, o lanzar acciones basadas en la localización como notificación push, etc.

iBeacon ofrece dos métodos de API para detectar dispositivos ibeacons. Ranging, que sólo funciona cuando la aplicación está activada y proporciona estimaciones de proximidad; Monitoring, que funciona incluso si la aplicación está en background, y proporciona información relativa a entrar y salir de la región definida por un beacon determinado.

Las tramas que envían los dispositivos constan de tres campos:

- **UUID** que identifica el beacon.
- **Major** es el número que identifica un subgrupo de beacons dentro de un grupo más grande.
- **Minor** número de identificación a un beacon específico.

La App asociada a los beacons, recibe el UUID, Major y Minor. Además, se utiliza la potencia de la señal recibida para estimar la distancia entre el beacon y el Smartphone. La App utiliza los tres campos para buscar en una base de datos local o remota la información que se debe mostrar al cliente.

3.4 iOS

iOS es un sistema operativo móvil de la multinacional Apple Inc. Originalmente desarrollado para el iPhone, después se ha usado en dispositivos como el iPod touch y el iPad.

Los elementos de control consisten de deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan a sacudir el dispositivo o al rotarlo en alguna dirección.

iOS se deriva de OS X, que a su vez está basado en Darwin BSD, y por lo tanto es un sistema operativo tipo Unix. iOS cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Principales", la capa de "Medios" y la capa de "Cocoa Touch".

La existencia del iPhone OS se remonta a enero de 2007 en la Macworld Conference & Expo, aunque el sistema no tuvo un nombre oficial hasta que salió la primera versión beta del iPhone SDK un año más tarde, en marzo de 2008. Antes de esto se consideraba simplemente que el iPhone ejecutaba OS X o una versión modificada de NewtonOS. A partir de entonces se llamaría iPhone OS. El lanzamiento oficial del iPhone OS tuvo lugar en junio de 2007.

El interés en el SDK aumentaría notablemente en los siguientes meses debido al aumentante crecimiento de la plataforma iPhone, que se vio incrementado en septiembre de 2007 por el lanzamiento del iPod Touch, un dispositivo con las capacidades multimedia del iPhone, pero sin la capacidad de hacer llamadas telefónicas solo por redes.

En enero de 2010 Steve Jobs, anunció el iPad, un dispositivo muy similar al iPod Touch, pero con un propósito orientado hacia la industria de contenidos. Este dispositivo, apoyado en una pantalla táctil de mayor dimensión, compartiría sistema operativo con sus dos exitosos hermanos, y vendría acompañado de una aplicación oficial para la compra y lectura de libros electrónicos, iBooks.

Tras más de 185000, las aplicaciones disponibles para iPhone OS a través de la App Store8, durante la presentación del iPhone 4, en Junio de 2010, Steve Jobs anunció que iPhone OS pasaría a ser llamado oficialmente como iOS.

En de 2013 es presentado iOS 7 en la WWDC 2013 a las 10:00 tiempo de San Francisco como "El mayor cambio de iOS desde el iPhone original", cambia por completo

el diseño gráfico del sistema, haciéndolo más plano y con nuevos íconos, trae nuevas características como AirDrop, Filtros de cámara, Fondo dinámico entre muchas otras. Ese mismo día se liberó la beta 1 para desarrolladores. En la misma conferencia se dieron a conocer los datos oficiales de iOS hasta la fecha, indicando que habían sido vendidos más de 600 millones de iDevices.

La última versión estable del sistema operativo a día de hoy es la versión iOS 10

3.4.1 Características

Los componentes principales del sistema operativo de iOS son los siguientes:

Pantalla principal: La pantalla principal (llamada «SpringBoard») es donde se ubican los iconos de las aplicaciones y el Dock en la parte inferior de la pantalla donde se pueden anclar aplicaciones de uso frecuente. Aparece al desbloquear el dispositivo o presionar el botón de inicio. La pantalla tiene una barra de estado en la parte superior para mostrar datos, tales como la hora, el nivel de batería, y la intensidad de la señal. El resto de la pantalla está dedicado a la aplicación actual.

Carpetas: Con iOS 4 se introdujo un sistema simple de carpetas en el sistema. Se puede mover una aplicación sobre otra y se creará una carpeta, y así se pueden agregar más aplicaciones a esta mediante el mismo procedimiento. Pueden entrar hasta 12 y 20 aplicaciones en el iPhone y iPad respectivamente. El título de la carpeta es seleccionado automáticamente por el tipo de aplicaciones dentro de ella, pero puede ser editado por el usuario.

Con la salida de iOS 7, la cantidad máxima de aplicaciones por carpeta aumentó considerablemente, pues al abrir una carpeta se muestran 9 iconos (3x3), y al agregar más aplicaciones se van creando páginas a las que se pueden acceder deslizando sobre la pantalla.

Seguridad: Antes de la salida de iOS 7 al mercado, existía un enorme índice de robos de los diversos modelos de iPhone, lo que provocó que el gobierno estadounidense solicitara a Apple diseñar un sistema de seguridad infalible que inutilizara los equipos en caso de robo. Fue creada entonces la activación por iCloud, la cual solicita los datos de acceso de la cuenta del usuario original, lo que permite bloquear e inutilizar el equipo al perderlo o ser víctima de robo del mismo. De igual manera, es posible conocer la ubicación vía GPS del dispositivo y mostrar mensajes en la pantalla. Hasta la fecha no existe un método comprobado para saltarse la activación de iCloud, lo que convierte a iOS 7+ en el SO móvil más seguro del mercado. Con la llegada de iOS 9.1, es imposible encontrar vulnerabilidades que afecten al dispositivo, llevándole a Apple, a ocupar el puesto más alto en la lista de los softwares más seguros.

Centro de notificaciones: Con la actualización iOS 5, el sistema de notificaciones se rediseñó por completo. Las notificaciones ahora se colocan en un área por la cual se puede acceder mediante un deslice desde la barra de estado hacia abajo. Al hacer un toque en una notificación el sistema abre la aplicación. La pantalla inicial de iOS contiene varias aplicaciones, algunas de las cuales están ocultas por defecto y pueden ser activadas por el usuario mediante la aplicación "Ajustes".

Todas las «utilidades», como Notas de Voz, Reloj, Brújula y Calculadora están en una carpeta llamada «Utilidades» desde la versión 4.0.12. Varias de las aplicaciones incluidas están diseñadas para trabajar juntas, permitiendo compartir datos de una aplicación a otra. (por ejemplo, un número de teléfono puede ser seleccionado desde un correo electrónico y guardarlo como un contacto o para hacer una llamada)

El iPod Touch tiene las mismas apps que están presentes en el iPhone, con excepción de Teléfono, Mensajes (aunque sí iMessage) y Brújula. Hasta iOS 5, en el iPhone y el iPad los iconos de música y vídeos estaban juntos en una sola aplicación, pero luego se separaron en 2, Música y Videos. Por defecto, en el dock del iPhone, ubicado en la parte inferior de la pantalla de inicio, están los iconos "estrella", que son Teléfono, Mail, Safari y Música. En el iPod touch, estos iconos son Música, Safari, Mail y iMessage.

El iPad también tiene las mismas aplicaciones que el iPhone, excluyendo Bolsa, Tiempo, Calculadora, Voice Memos, Teléfono, Mensajes (aunque si iMessage) y Nike+iPod, apps separadas para música y vídeo igualmente se usan (como en el iPhone). Varias apps por defecto están reescritas para tomar ventaja de la pantalla más grande. El dock por defecto incluye Safari, Mail, Fotos y Música. Desde iOS 6 en adelante, el iPad 2 y iPad 3 tienen la aplicación de Reloj.

Multitarea opcional: Antes de iOS 4, la multitarea estaba reservada para aplicaciones por defecto del sistema. A Apple le preocupaba los problemas de batería y rendimiento si se permitiese correr varias aplicaciones de terceros al mismo tiempo. La multitarea sólo era compatible desde el iPhone 3GS, iPad 1 y el iPod Touch (3.ª generación). A partir de iOS 4, dispositivos de tercera generación y posteriores permiten el uso de APIs para multitarea, específicamente:

- Audio en segundo plano
- Voz IP
- Localización en segundo plano
- Notificaciones push
- Notificaciones locales
- Completado de tareas

3.4.2 Kit de desarrollo software

El 17 de octubre de 2007, Steve Jobs anunció un Kit de desarrollo de software o SDK, que estaría disponible para terceros y desarrolladores en febrero del 2008. El SDK fue liberado finalmente el 6 de marzo de 2008, permitiendo así a los desarrolladores hacer aplicaciones para el iPhone y iPod Touch, así como probarlas en el "iPhone simulator". De cualquier manera, y hasta la llegada de xcode 7 solo era posible probar las apps en los dispositivos después de pagar la cuota del iPhone Developer Program. A partir de xcode 7 es posible utilizar un dispositivo iOS para probar las aplicaciones sin necesidad de cuenta de desarrollador.

El lenguaje de programación utilizado para desarrollar las aplicaciones iOS y Mac OS era Objective-C, pero desde hace unos pocos años se trabaja con Swift, que es el nuevo lenguaje de programación definido por Apple para programar sus dispositivos. No obstante, todavía se puede escribir código fuente usando el lenguaje de programación Objective-C.

Los desarrolladores pueden poner un precio por encima del mínimo (0,99 dólares) a sus aplicaciones para distribuirlas en el App Store, de donde recibirán el 70 % del dinero que produzca la aplicación. En alternativa, el desarrollador puede optar por lanzar la aplicación gratis, y de esta forma no pagar ningún costo por distribuir la aplicación (excepto por la cuota de la membresía).

3.4.3 X-Code

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se puede descargar e instalar gratuitamente para Mac OS X. Xcode trabaja conjuntamente con Interface Builder, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Swift, Objective-C, Objective-C++, Java y AppleScript, mediante una amplia gama de modelos de programación, incluyendo, pero no limitado aCocoa, Carbón y Java. Otras compañías han añadido soporte para GNU Pascal, Free Pascal, Ada y Perl.

Entre las características más apreciadas de Xcode está la tecnología para distribuir el proceso de construcción a partir de código fuente entre varios ordenadores, utilizando Bonjour.

En relación a la historia de Xcode, se introdujo el 24 de octubre de 2003 junto con la versión 10.3 de Mac OS X, siendo desarrollado a partir del anterior entorno de

desarrollo, Project Builder, al que sustituyó. Project Builder, a su vez, también era una herencia de la compañía NeXT, fusionada con Apple en 1996.

La aparición de Xcode 2.1 en junio de 2005 fue significativa porque proporcionó a la comunidad de desarrolladores las herramientas para crear binarios universales que permiten al software creado para Mac OS X ser ejecutado tanto en la arquitectura PowerPC como en la nueva, basada en Intel (x86). Esta versión integró además las herramientas y marcos de trabajo WebObjects de Apple para construir aplicaciones y servicios web de Java, que anteriormente se vendían como un producto separado por un precio de 699 $.

Con el lanzamiento de Mac OS X v10.5 también lo fue el Xcode 3.0, que tenía como principales novedades la inclusión de Objetive-C 2.0, un nuevo Interface Builder, la opción de refactorizar proyectos y hacer "snapshosts" del proyecto entre otras.

Xcode 4, lanzado a principios de 2011, incluía como novedades una nueva interfaz y la compatibilidad con Mac OS X 10.7 Lion. Con esta versión, Xcode dejó de ser compatible con Mac OS X 10.5 Leopard.

3.5 Raspberry Pi 2 modelo B

3.5.1 Vista conjunta

La Raspberry Pi 2 modelo B (Ilustración 3.3), como ya se expuso en el capítulo anterior, es un mini-PC que cuenta con un procesador de 900 MHz quad-core ARM Cortex A7 montado en un socket BCM2837, un núcleo gráfico de procesos con cuatro GPUs y 1 GB de RAM. Dispone de cuatro conexiones USB 2.0, puerto Ethernet, conector de audio tipo Jack de 3.5 mm y HDMI, y es capaz de reproducir vídeo de calidad BluRay utilizando H.264 a 40 Mbits/s, gracias a la aceleración 3D aportada por las librerías OpenGL ES2.0 y OpenVG.

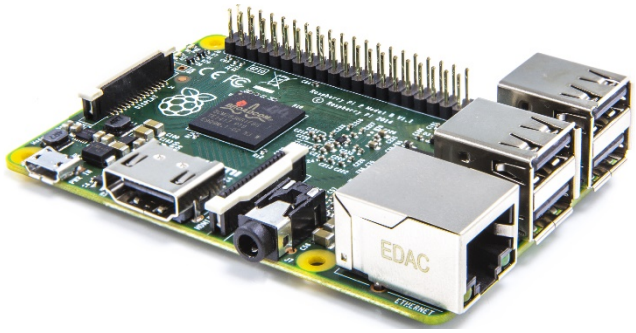


Ilustración 3.3.- Vista de la Raspberry Pi 2 modelo B

Su capacidad de conexión, indispensable en la mayoría de proyectos, es de 40 GPIOs digitales de uso general.

3.5.2 Resumen.

Especificaciones:
Procesador: Quad-Core Cortex A7 a 900 MHz
RAM: 1GB.
Puertos:
4 x USB 2.0
1 x 40 GPIO pin
1 X HDMI
1 x Ethernet
1 x Combo audio/mic
1 x Interfaz de cámara (CSI)
1 X Interfaz de Pantalla (DSI)
1 x Micro SD
1 x Núcleo Grafico 3D

Tabla 3.1.- Resumen de las características de la placa Raspberry Pi

3.5.3 Memoria

Raspberry Pi B cuenta con 1 GB de memoria RAM, hasta 32 GB de memoria flash con tarjeta SD y para almacenamiento, además de que siempre es posible conectar por USB dispositivos de mayor capacidad.

Es importante tener en cuenta que como el software se instala en la tarjeta SD, influye mucho la clase de dicha tarjeta, ya que ésta a su vez determina la velocidad de transferencia, que se traduce finalmente en un mejor rendimiento de la Raspberry Pi. Cabe mencionar que no todas las tarjetas SD existentes en el mercado son compatibles con este dispositivo y existen listados donde comprobar que un determinado modelo y tamaño es compatible con el sistema. Un ejemplo de este tipo de listas es: http://elinux.org/RPi_SD_cards.

3.5.4 GPIO

La SBC Raspberry Pi dispone de varias GPIOs (*General Purpose Input/Output)*, tal y como se muestra en la Ilustración 3.4.

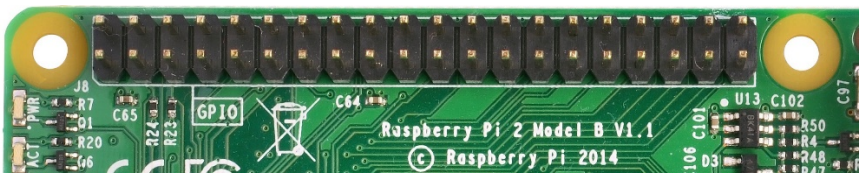


Ilustración 3.4.- Vista del GPIO de Rasbperry Pi 2 modelo B

En la Ilustración 3.5 se muestra la configuración de cada pin mediante un código de colores. Los pines de tipo *GPIO* mostrados se emplean en la entrada y salida digital.

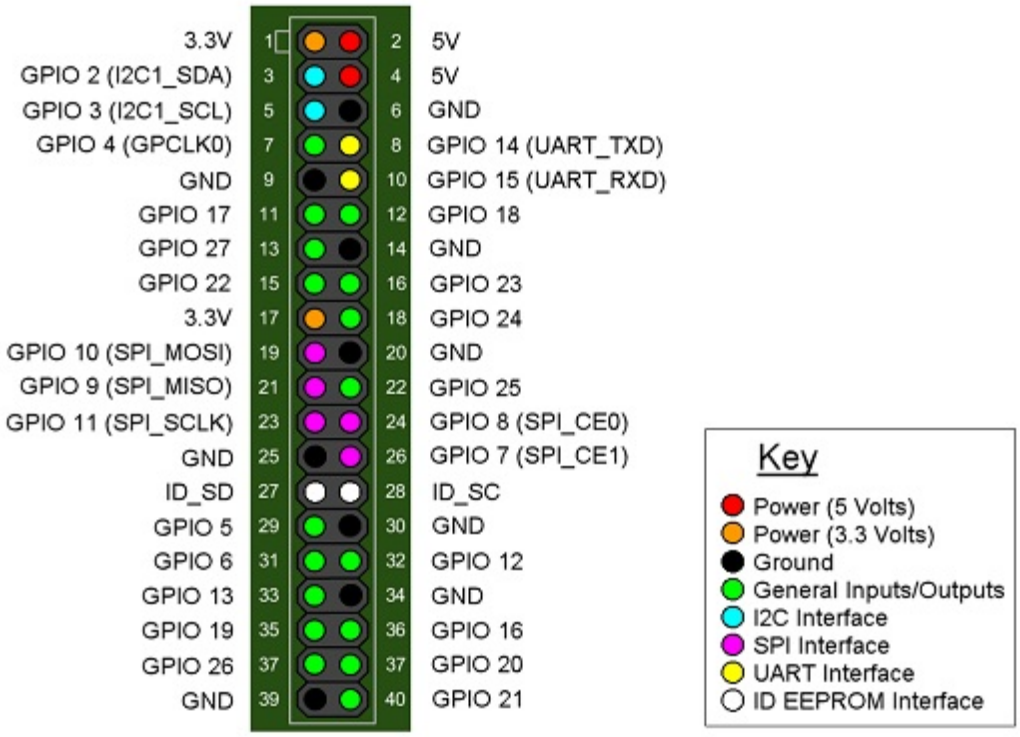


Ilustración 3.5.- Configuración de los pines del GPIO

3.5.5 Riesgos de sobrecarga

Raspberry Pi posee diferentes entradas y salidas al sistema y es conveniente conocer cuáles son los límites que se pueden alcanzar en su funcionamiento.

- La alimentación del sistema Raspberry se realiza a través de micro-USB, de forma que su tensión de alimentación debe ser de 5 V, admitiendo una tolerancia de ± 0,25 V. Si se sobrepasa este valor el sistema viene preparado con un fusible de protección de 1,1 A.

- Los USB disponibles permiten la conexión de otros dispositivos externos, sin embargo, hay que controlar la intensidad que estará aportando dado Raspberry Pi 2 modelo B ya que no está limitada. Así, la intensidad que se puede demandar a los USB depende de la intensidad con la que se alimenta Raspberry Pi. Esto hace que el límite teórico sea de 1,1 A, que es el máximo permitido por el fusible menos la intensidad demandada por la placa. Hay que tener en cuenta que dispositivos de gran impedancia pueden hacer que se reinicie el dispositivo por caída de tensión al conectarlos.

- El conector GPIO posee unos pines que de forma continuada aportan 5 V y 3,3 V. Los pines de 5 V permiten un consumo acorde con la alimentación del dispositivo (al igual que las salidas USB). En el caso de los pines de 3,3 V no debe conectarse nada que haga que la intensidad supere 50 mA. En general como máximo, con una fuente normal de alimentación, no se recomienda superar los 300 mA.

- Los pines I/O digitales no deben tener conectados nada que consuma más de 16 mA de intensidad con un total de 50 mA entre todos los pines.

Como curiosidad, cabe destacar que en los USB del modelo A o Rev1 estaba limitada la intensidad de salida mediante fusibles a 140 mA. Estos fusibles resultan tener una resistencia de 5 Ω, ocasionando que la tensión en el USB sea de 4,5V, traduciéndose en un mal funcionamiento de algunos dispositivos externos. Por ello, se decidió eliminar esta protección en el modelo B.

3.5.6 Programación

Dado que la mayoría de sistemas operativos están basados en Linux, la programación de la Raspberry Pi se realiza mediante cualquier programa válido para Linux siempre y cuando exista el compilador adecuado para arquitectura ARM.

De forma general, la programación trae consigo la interacción con GPIO y ésta se puede realizar de varias formas y utilizando diferentes lenguajes de programación de alto nivel, aquí se muestran algunas:

- Con la línea de comandos. El acceso a los pines del GPIO se realiza con comandos de la propia línea de Linux en tres pasos. Primero se selecciona el pin a utilizar mediante el comando *echo [nº_pin] > /sys/class/gpio/export*. En segundo lugar, se señala si el pin es de salida o entrada con el comando *echo [in/out] > /sys/class/gpio/gpio17/direction*. Por último, se señala el valor del pin. En el caso de que se haya seleccionado que es de salida con *echo 0 > /sys/class/gpio/gpio17*/value o se recoge si se seleccionó de entrada.

- Utilizando Python. En este caso se emplean funciones específicas de una librería para python llamada RPi.GPIO. Para seleccionar que un pin será de entrada o salida se utilzará la función *GPIO.setup (17, GPIO.[IN/OUT])*. Para seleccionar un valor o leerlo se utilizarán respectivamente *GPIO.output() y GPIO.input()*.

- Mediante una herramienta de hardware de Matlab. Matlab posee una herramienta que permite la interacción con el GPIO mediante sencillas funciones. Para señalar que un pin será de entrada o salida se utiliza *configureDigitalPin(mypi,4,'input/output')* y para leer o escribir un valor se usan las funciones *readDigitalPin()* y *writeDigitalPin()*.

3.5.7 Software

Como ya se mencionó en el capítulo anterior, hay disponibles varios sistemas operativos para Raspberry Pi, sin embargo, aquí se van a detallar las características de *Raspbian*, ya que es el software que se va emplear en el presente proyecto. Se ha seleccionado dicho sistema operativo por su uso extendido y su gran versatilidad.

Raspbian es un sistema operativo gratuito basado en Debian Wheezy y optimizado para el hardware de Raspberry Pi. Sin embargo, aporta mucho más que un sistema operativo puro, ya que cuenta con más de 35000 paquetes de software pre-compilado, que hace más sencilla la instalación en el dispositivo. Desde su primera versión en junio de 2012, Raspbian está en continuo desarrollo aumentando el citado número de paquetes y haciéndose cada vez más estable.

En diciembre de 2012, junto a la versión 2012-12-16-wheezy-raspbian de Raspbian, se lanzó la tienda de aplicaciones *Pi Store*, que en el momento de salida incluía aplicaciones como *LibreOffice* o *Asterisk*. En esta plataforma se puede poner a disposición de todos los usuarios de Raspbian contenidos gratuitos o de pago, como archivos binarios, código python, imágenes, audio o vídeo.

La distribución usa *LXDE* como escritorio y *Midori* como navegador web. Además, contiene herramientas de desarrollo como *IDLE* para el lenguaje de programación *Python* o *Scratch*.

Destaca también el menú *raspi-config,* que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente. Entre sus funciones, permite expandir la partición root para que ocupe toda la tarjeta de memoria, configurar el teclado, aplicar overclock, etc.

Al igual que ocurre con cualquier versión de Linux, posee dos posibles interfaces de usuario. Una línea de comandos y una interfaz gráfica accesible mediante el comando *startx*. En la Ilustración 3.6 aparece la interfaz gráfica que por defecto muestra Raspbian tras su instalación.

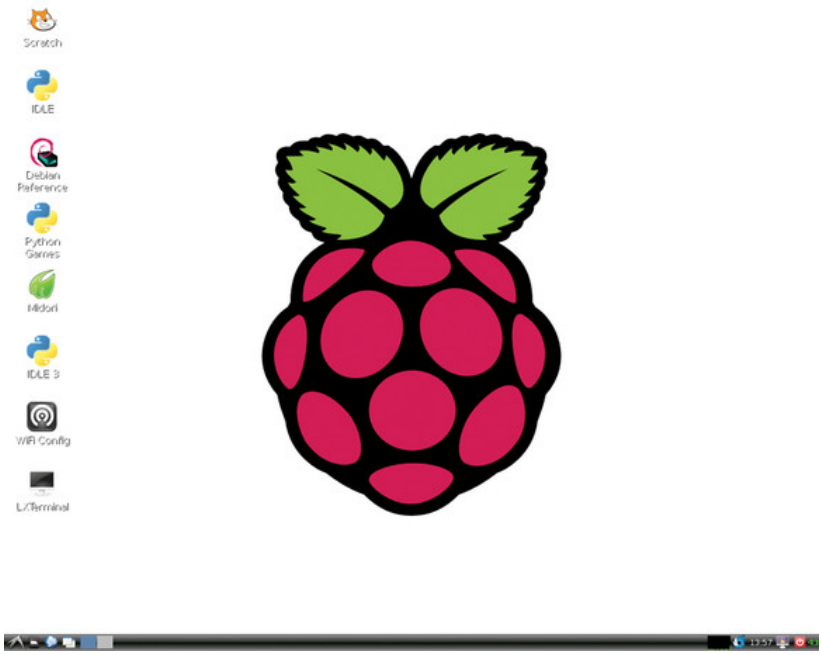


Ilustración 3.6.- Vista interfaz gráfica de Raspbian

Toda la información, novedades, ayudas para instalación, foros y mucho más se puede encontrar en la web de Raspbian: http://raspbian.org/

3.5.7.1 Instalación de Raspbian

La instalación de Raspbian se realiza en una tarjeta SD. Para obtener la imagen de la distribución existen varias URL disponibles, pero es recomendable utilizar la oficial: http://www.raspberrypi.org/downloads/

En primer lugar se debe cargar la imagen descargada en la tarjeta SD. Para ello, se puede utilizar algún programa disponible. Uno gratuito y open-source disponible es *Win32 Disk Imager* compatible con Windows. Una vez conectada la SD al PC, se abre el programa y se indica que se va a copiar en la tarjeta y se selecciona el directorio con la imagen de Raspbian descargada. Este proceso se puede ver en la Ilustración 3.7.

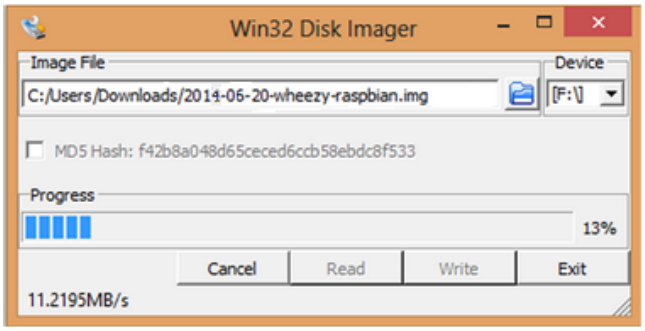


Ilustración 3.7.- Preparación de la tarjeta SD para instalación de Raspbian

Una vez finalizado el proceso, se introduce la tarjeta en la Raspberry Pi y se inicia automáticamente el proceso de instalación. Una vez finalizado, se pueden realizar algunos ajustes, tales como seleccionar idioma, el idioma del teclado y el tamaño de partición se quiere asignar al SO, entre otros (ver Ilustración 3.8), gracias al ya citado menú Raspi-config.

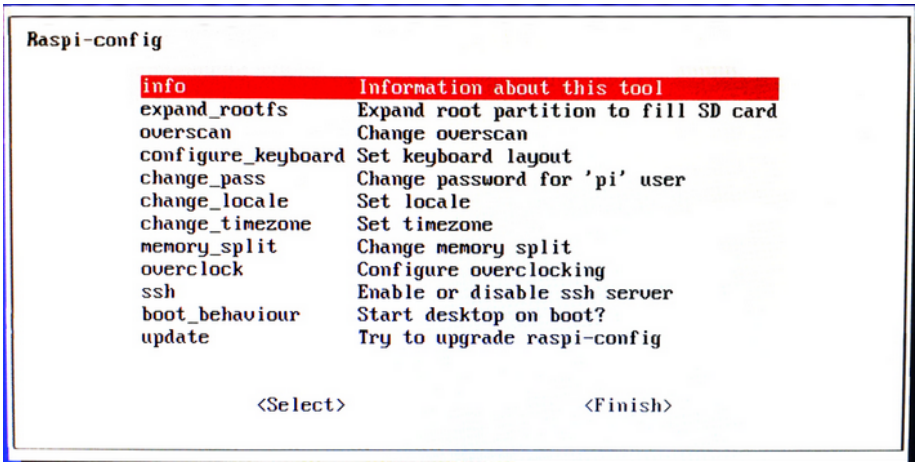


Ilustración 3.8.- Ajustes disponibles tras la instalación

3.6 ESP8266

El ESP8266 es un chip Wi-FI de bajo coste y soporte completo de la pila TCP/IP y con capacidad de microcontrolador producido por el fabricante chino con sede en Shanghai, Sistemas Espressif.

El chip por primera vez llamó la atención de los fabricantes occidentales en agosto de 2014, con el módulo ESP-01, fabricado por un fabricante third-party, AI-Thinker. Este pequeño módulo permite a los microcontroladores conectarse a una red Wi-Fi y hacer conexiones TCP/IP simples usando comandos estilo Hayes. Sin embargo, en ese momento no había casi ninguna documentación en inglés sobre el chip y los comandos que aceptaba. El bajo precio y el hecho de que había muy pocos componentes externos en el módulo que sugirió que podría llegar a ser muy barato, lo que atrajo a muchos hackers para explorar el módulo, el chip, y el software en el mismo, así como para traducir la documentación en chino.

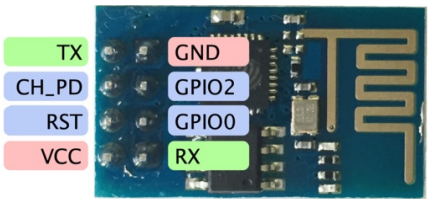


Ilustración 3.9.- Esquema de un ESP8266

3.6.1 Características

- 32-bit RISC CPU: Tensilica Xtensa LX106 funcionando a 80 MHz
- 64 KiB de RAM instrucción, 96 KiB de RAM de datos
- QSPI externa de flash - 512 KiB de MIB 4 (hasta 16 MiB es compatible)
- IEEE 802.11 b / g / n Wi-Fi
- Integra interruptor TR, balun, LNA, amplificador de potencia y la red de adaptación
- WEP o WPA / WPA2 redes de autenticación, o abiertas
- 16 GPIOs disponibles
- Soporte con buses digitales SPI , I²C, entre otros.
- I²S interfaces con DMA (pines de intercambio con GPIO).
- UART en los pines dedicados, además de una transmisión de sólo UART se puede habilitar en la línea GPIO2
- ADCs de 10 bits de resolución

Tanto la CPU y velocidades de reloj de flash se pueden doblar por overclock en algunos dispositivos. Concretamente, la CPU puede funcionar a 160 MHz y la flash se puede acelerar desde 40 MHz a 80 MHz. El éxito varía de chip a chip.

3.6.2 SDK

A finales de octubre de 2014, Espressif dio a conocer un kit de desarrollo de software (SDK) que permitió programar el chip, eliminando la necesidad de un microcontrolador separado. Desde entonces, ha habido muchos lanzamientos oficiales SDK de Espressif. Concretamente, Espressif mantiene dos versiones del SDK, uno que se basa en RTOS y el otro basado en devoluciones de llamada.

También hay alternativas al SDK oficial de Espressif. Algunas de ellas son las siguientes:

- esp-abierta-SDK, que se basa en la cadena de herramientas GCC.
- "Kit de desarrollo no oficial" por Mikhail Grigorev.
- NodeMCU : un firmware basado en Lua.
- Arduino: un C ++ firmware basado. Este núcleo permite al ESP8266 CPU y sus componentes Wi-Fi para ser programado como cualquier otro dispositivo Arduino. El ESP8266 Arduino Core está disponible a través de GitHub .

- MicroPython: un "port" de la MicroPython (una implementación de Python para dispositivos embebidos) a la plataforma ESP8266.

- ESP8266 BÁSICO: Una fuente abierta con un intérprete básico adaptado específicamente para el Internet de las cosas.

Capítulo 4

Caso de estudio. Descripción del hardware y software desarrollado.

4.1 Introducción.

Tras la descripción realizada en los capítulos anteriores, ya es posible llevar a cabo la implementación de un caso de estudio concreto que muestre el desempeño de la aplicación y el sistema de iluminación inteligente. En primer lugar, se detallará el caso de estudio de forma general, comentando su funcionamiento y objetivos. Posteriormente, se describirá el hardware y software de forma más específica.

El estudio, en líneas generales, consiste en la implementación de un sistema de iluminación inteligente que haga más eficiente la tarea de uno convencional.

La iluminación inteligente surge como aplicación de las nuevas tecnologías de entornos inteligentes. Con el sistema propuesto será posible que la iluminación de cualquier sala responda automáticamente a la presencia humana, a la vez que conseguir una optimización de la energía eléctrica consumida.

Esta aplicación está pensada tanto para espacios reducidos como amplios, tanto de trabajo como de ocio, habiéndose desarrollado como ejemplo un caso único, pero pudiendo fácilmente escalar el número de dispositivos.

4.2 Descripción de la arquitectura

El caso de estudio consta de una bombilla que se enciende mediante la App desarrollada. En concreto, cuando la App detecta el dispositivo beacon se conectará con el bróker (RPi) mediante mensajes MQTT. El siguiente paso consiste en que en bróker intercambiará información con el ESP8266 (controlador de la bombilla mediante relé) mediante una red WiFi creada por el punto de acceso. En la Ilustración 4.1 se muestra una fotografía del sistema en la que se pueden identificar los elementos mencionados anteriormente.

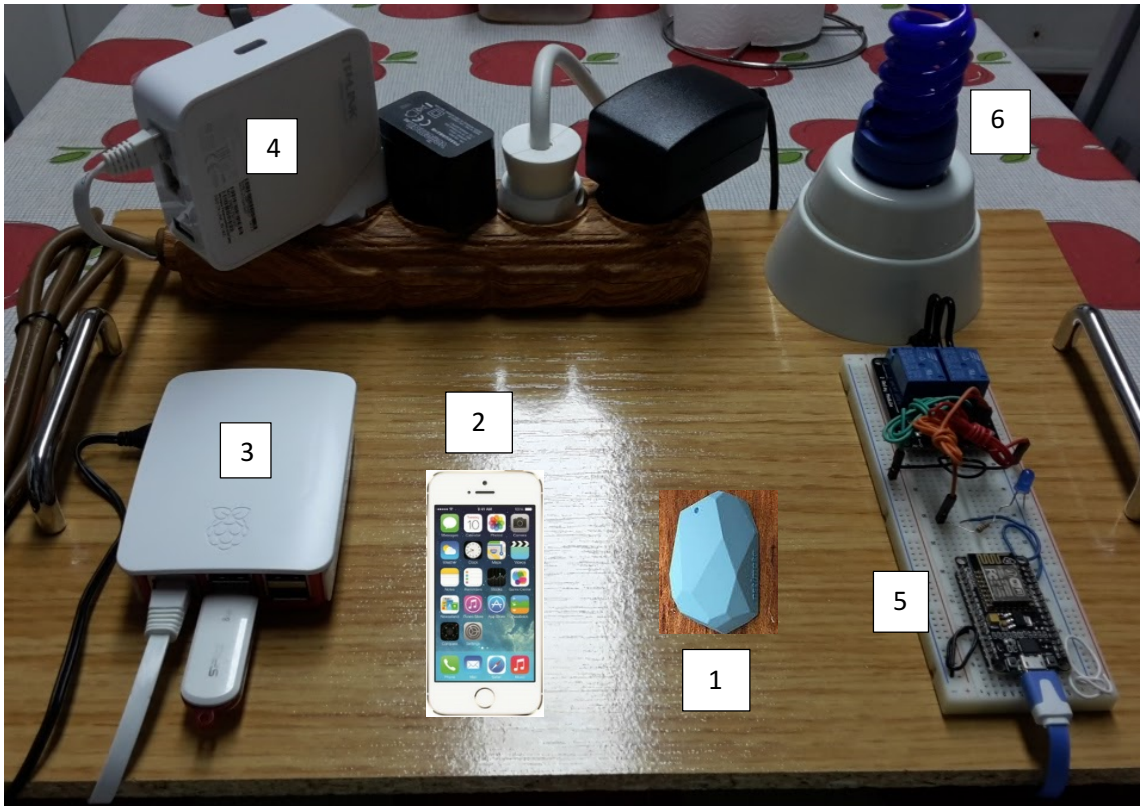


Ilustración 4.12 Caso de estudio (Layout del proyecto)

De esta forma, los dispositivos que componen el sistema son en definitiva los enumerados a continuación:

- Dispositivo 1. El beacon, que emite periódicamente mensajes con el UUID, el major y el minor.

- Dispositivo 2. El Smartphone, que ejecuta la App desarrollada y recibe los mensajes enviados por el beacon. De esta manera se puede saber si se ha entrado o salido en la zona que identifica el dispositivo beacon. Por tanto, en el caso de estar la luz apagada y entrar en la zona del mismo, se encendería la luz.

- Dispositivo 3. El bróker, basado en el SBC Raspberry Pi y sirve de puente entre el controlador de la lámpara (ESP8266) y el dispositivo móvil inteligente.

- Dispositivo 4. AP, o punto de acceso, el cual creará la Red WiFi que permite conectar los módulos ESP8266 con el bróker.

- Dispositivos 5. El módulo WiFi ESP8266, que habiéndose programado en Arduino, recibirá las órdenes del bróker para encender o apagar las luces

- Dispositivo 6. La luz, que, en este caso, es una bombilla de bajo consumo.

A continuación, se describirán en profundidad cada uno de los dispositivos enumerados.

4.3 Dispositivo 1. Beacon.

El dispositivo 1, tal y como se ha comentado anteriormente es el encargado de enviar periódicamente mensajes que incluyen los campos UUID, major y minor, tal y como está especificado en la descripción de la tecnología iBeacon.

4.3.1 Hardware.

A nivel hardware el beacon seleccionado de la compañía Estimote incluye los siguientes elementos:

- Elemento de cómputo de 32-bits ARM® Cortex y módulo de comunicación BLE 4.0 (2,4 GHz)
- Pila de botón CR2477
- Acelerómetro
- Sensor de temperatura

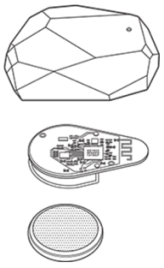


Ilustración 4.2 Esquema de un Beacon

4.3.2 Vista del dispositivo real.

En la Ilustración 4.3 se muestra la imagen real del dispositivo 1, en la que se puede observar la batería, la PCB del dispositivo y la carcasa fabricada flexible y estanca del dispositivo.

Ilustración 4.3. Vista real del dispositivo 1

4.3.3 Software.

El software principalmente deriva del Smartphone que recibe información de los *beacons*, pero si hay una pequeña configuración que se puede realizar a través de la página Web de *Estimote*.

En la página hay un primer menú para seleccionar de los *beacons* adquiridos cual se quiere configurar.

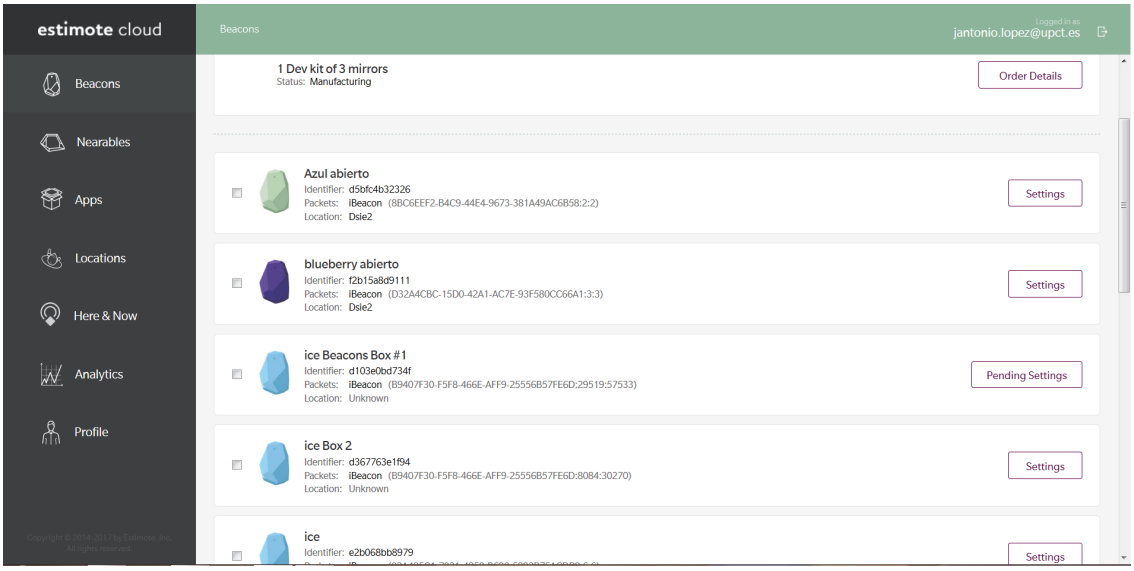


Ilustración 4.4. Menú de *beacons* a configurar

Pulsando sobre el botón "Settings" del dispositivo deseado aparecerá la siguiente pantalla:

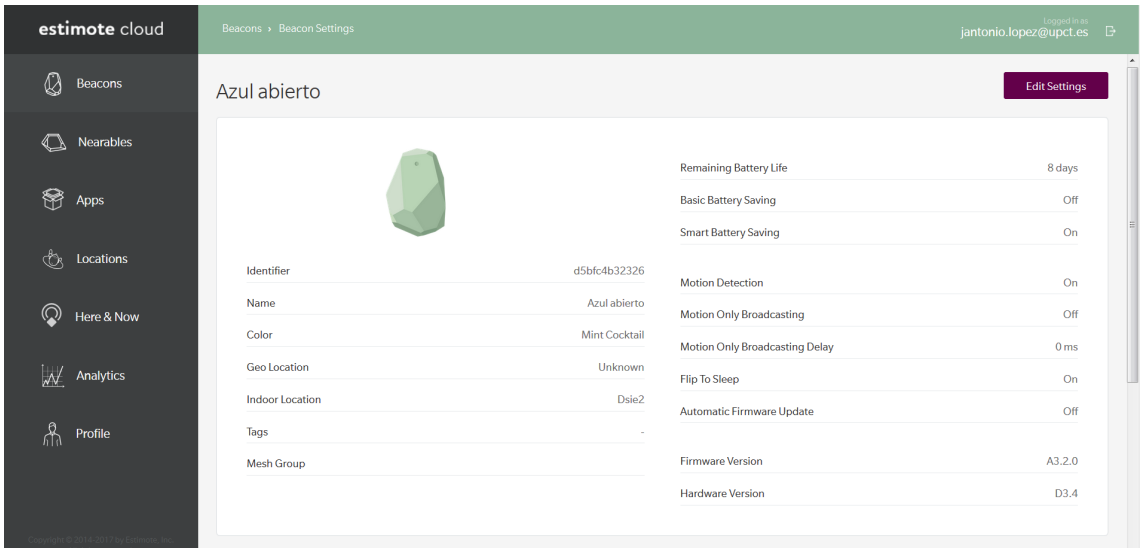


Ilustración 4.5. Configuración del *beacon*

En la configuración se encuentra el número de identificación del beacon, su nombre, el color, la localización tanto geográfica como interna, opciones de consumo de energía, como uso inteligente de la batería, la configuración de su sensor de movimiento, el cual incluye la opción *flip to sleep,* la cual es interesante y es tan simple como que permitirá que al voltear el *beacon* boca abajo este se apague, así como opciones del firmware y la versión del hardware.

Además de las opciones descritas anteriormente, se pueden encontrar las siguientes opciones, que son sumamente importantes a la hora de trabajar con la tecnología iBeacon.

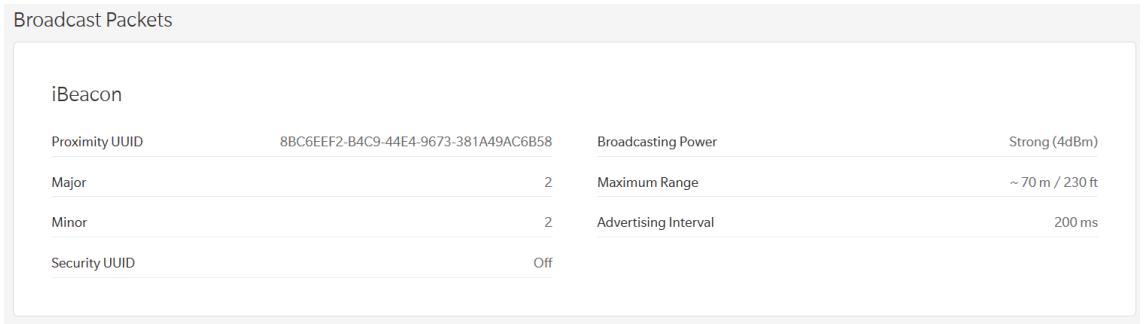


Ilustración 4.6. Configuración del broadcasting del *beacon*

En concreto, se muestran el UUID, el major y el minor del *beacon*, parámetros que son fundamentales si se quiere programar una App con estos dispositivos. También las opciones de comunicación, tales como la potencia de transmisión, el alcance de la

comunicación, la frecuencia con la que se envían los valores anteriores, así como la configuración del nivel de seguridad.

4.4 Dispositivo 2. Smartphone

El dispositivo 2 es el Smartphone, que ejecuta la App que detecta que se ha entrado o salido de la zona definida por el beacon (modo monitoring), y en caso de ser necesario encenderá o apagará la luz mediante el intercambio de información el módulo ESP8266 a través del bróker.

4.4.1 Hardware.

En cuanto a hardware del Smartphone hay un mínimo de características requeridas, principalmente que el terminal tenga conectividad BLE y WiFi. Para este trabajo se usó un iPhone 4S que cumple con las características mencionadas.

4.4.2 Vista real del dispositivo.

En la Ilustración 4.7 se muestra el dispositivo 2 utilizado, el iPhone 4S. Naturalmente el sistema desarrollado sería compatible con dicho iPhone o superior, en términos de la App desarrollada.

Ilustración 4.7. Vista real del dispositivo 2.

4.4.3 Software.

La App se ha realizado con el entorno de desarrollo Xcode versión 7.3, que permite realizar aplicaciones compatibles con iOS 9.3. La aplicación está escrita en lenguaje Swift, pero como también se ha utilizado una biblioteca escrita con el lenguaje de programación

antiguo (Objective-C), ha sido necesario utilizar un fichero de puente entre ambos lenguajes, que será explicado posteriormente.

Como cualquier App para iOS, el primer paso habitual es definir la interfaz de la misma en el fichero con extensión "storyboard". En este caso se trata de una vista muy sencilla que incluye 4 botones y una etiqueta.

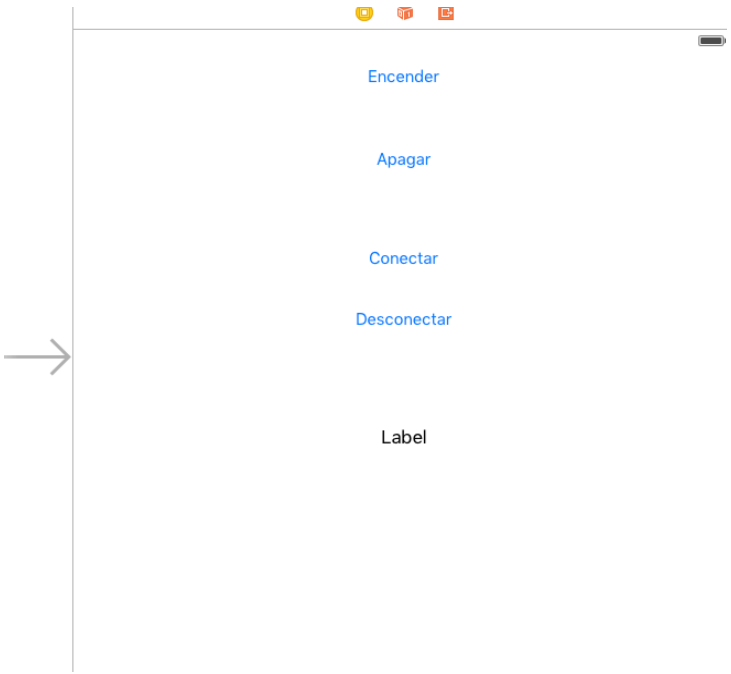


Ilustración 4.8. Interfaz de usuario: Main.storyboard

La estructura del proyecto se muestra en la siguiente ilustración, en la que se pueden identificar los ficheros fuente de la App, así como las 2 bibliotecas usadas: la primera de ellas para trabajar con los beacons y la segunda para intercambiar mensajes mediante MQTT.

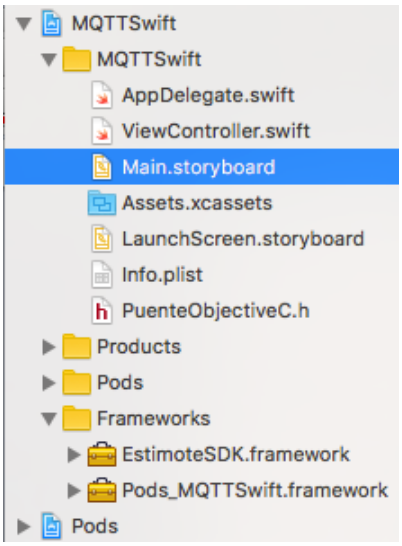


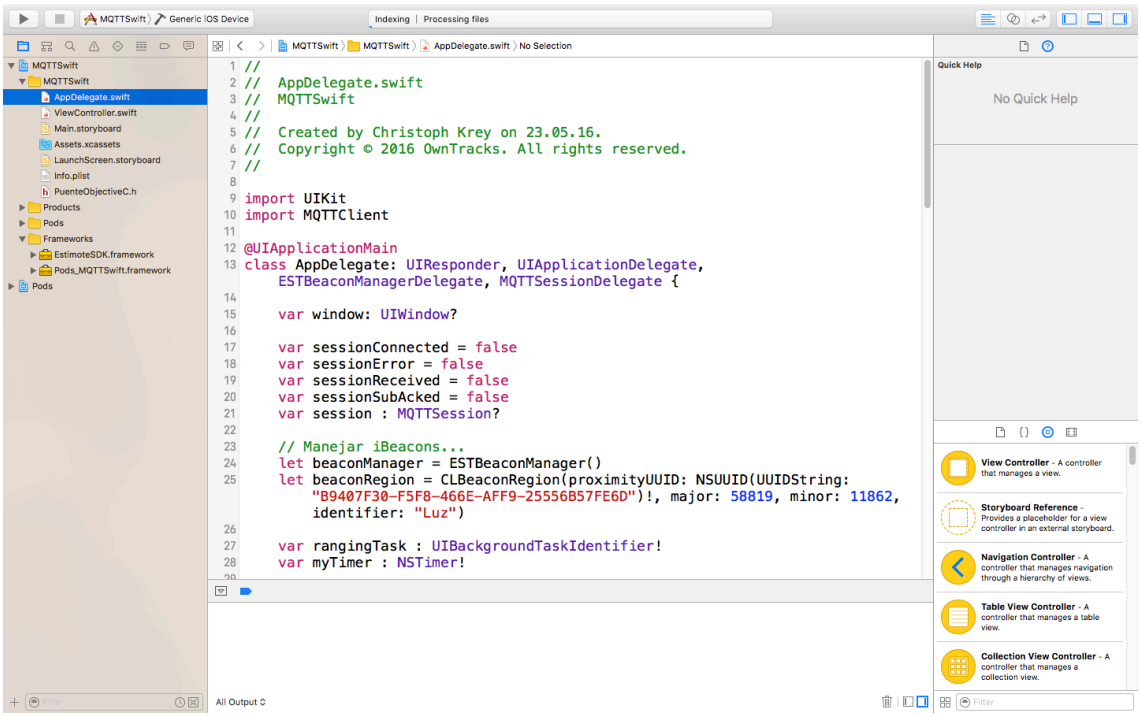
Ilustración 4.9. Estructura del proyecto

Como la biblioteca "EstimoteSDK.framework" está escrita en lenguaje Objective-C, ha sido necesario definir el fichero "PuenteObjectiveC.h", además de la correspondiente configuración en el fichero "Info.plist", tal y como está explicado en la página Web de Estimote.

El desarrollo para iOS sigue el paradigma MVC (Modelo-Vista-Controlador), de manera que la vista activa tiene asociada una clase controladora. En este caso la aplicación sólo tiene una vista (ver Ilustración 4.8) y la clase controladora de la misma está escrita en el fichero "ViewController.swift". En este punto hay que resaltar que todo el código escrito dentro de esta clase controladora sólo se ejecutaría cuando la App esté activa, así como la vista correspondiente. En este proyecto interesa que la aplicación pueda funcionar en segundo plano, aunque la App está cerrada. Por ello, el código principal de la App está escrito en el fichero "AppDelegate.swift".

En las líneas 17-21 se han definido un conjunto de variables para manejar la comunicación MQTT. Del mismo modo, en las líneas 24 y 25 se ha definido una variable para disponer de un manejador de los beacons, así como de la región, que tiene los mismo parámetros (UUID, major y minor) que el dispositivo beacon utilizado.

Por otro lado, iOS es muy estricto con la ejecución en segundo plano y sólo permite procesamientos muy ligeros y reducidos en tiempo. En este caso para ampliar el tiempo necesario de procesamiento se han definido tres variables necesarias: tarea en background, para manejar el temporizador necesario y una variable para almacenar una lista de beacons encontrados al realizar una búsqueda.




```
1  //
2  //  AppDelegate.swift
3  //  MQTTSwift
4  //
5  //  Created by Christoph Krey on 23.05.16.
6  //  Copyright © 2016 OwnTracks. All rights reserved.
7  //
8
9  import UIKit
10 import MQTTClient
11
12 @UIApplicationMain
13 class AppDelegate: UIResponder, UIApplicationDelegate,
       ESTBeaconManagerDelegate, MQTTSessionDelegate {
14
15     var window: UIWindow?
16
17     var sessionConnected = false
18     var sessionError = false
19     var sessionReceived = false
20     var sessionSubAcked = false
21     var session : MQTTSession?
22
23     // Manejar iBeacons...
24     let beaconManager = ESTBeaconManager()
25     let beaconRegion = CLBeaconRegion(proximityUUID: NSUUID(UUIDString:
           "B9407F30-F5F8-466E-AFF9-25556B57FE6D")!, major: 58819, minor: 11862,
           identifier: "Luz")
26
27     var rangingTask : UIBackgroundTaskIdentifier!
28     var myTimer : NSTimer!
```

Ilustración 4.10. Captura completa del entorno

Cuando se abre la App se ejecuta el método "didFinishLaunchingWithOptions", que inicializar el beacon manager, habilita la recepción de notificaciones y comienza a monitorizar la zona definida anteriormente, es decir, la aplicación detectará si entra o sale de la zona definida por el beacon, con el que estamos trabajando en modo monitoring. Finalmente, se llama a la función "requestStateForRegion", que permite conocer inicialmente si estamos dentro o fuera de la región definida por el beacon.

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
         // Override point for customization after application launch.
        self.beaconManager.delegate = self
        self.beaconManager.requestAlwaysAuthorization()
        UIApplication.sharedApplication().registerUserNotificationSettings(
                   UIUserNotificationSettings(forTypes: .Sound, categories: nil))
        self.beaconManager.startMonitoringForRegion(beaconRegion)

        //didEnterRegion se dispara si estamos fuera y entramos
        //de este modo podemos saber el estado inicial
        self.beaconManager.requestStateForRegion(beaconRegion)
        print("Estimote SDK ready!!!")
        return true
}
```

El siguiente método se ejecuta cada vez que se produce un cambio de estado en relación con la zona, y en este caso se ha utilizado realizando llamadas a print para depurar la App.

```
func beaconManager(manager: AnyObject, didDetermineState state: CLRegionState, forRegion
region: CLBeaconRegion) {

        if (state == CLRegionState.Inside){
            print("Inside...")
        }
        else if (state == CLRegionState.Outside){
            print("Outside...")
        }
        else if (state == CLRegionState.Unknown){
            print("Unkhown...")
        }
}
```

El siguiente método se ejecuta cada vez se entra en la región definida por el beacon. En este caso se llama a la función "EncenderApagar" con el valor para encender y se crea la tarea en segundo plano y el timer para asegurar el envío del mensaje por MQTT.

```
//HEMOS ENTRADO EN LA REGIÓN...
func beaconManager(manager: AnyObject, didEnterRegion region: CLBeaconRegion) {
         print("--> Begin didEnterRegion")
         EncenderApagar(true)
         rangingTask =
UIApplication.sharedApplication().beginBackgroundTaskWithExpirationHandler({() -> Void
in})
myTimer = NSTimer.scheduledTimerWithTimeInterval(10, target: self, selector:
#selector(AppDelegate.myTimerEvent), userInfo: nil, repeats: false)
        print("--> End didEnterRegion")
}
```

La función "EncenderApagar" inicializa la sesión MQTT, publica el valor en el topic necesario y libera la sesión MQTT.

```
func EncenderApagar(estado:Bool){
         self.session = MQTTSession();
         self.session!.delegate = self;
         self.session!.connectToHost("192.168.1.42",port:1883,usingSSL: false);
         while !self.sessionConnected && !self.sessionError {
            NSRunLoop.currentRunLoop().runUntilDate(NSDate(timeIntervalSinceNow: 1))
         }
```

```
            print("MQTT init!!!")
            if (self.sessionConnected){
               if(estado){
                   self.session!.publishData("0".dataUsingEncoding(NSUTF8StringEncoding,
allowLossyConversion: false),onTopic: "zona1/izquierda/",retain: false,qos: .AtMostOnce)
                   print("Se enciende...")
               }
               else{
                   self.session!.publishData("1".dataUsingEncoding(NSUTF8StringEncoding,
allowLossyConversion: false),onTopic: "zona1/izquierda/",retain: false,qos: .AtMostOnce)
                   print("Se apaga...")
               }
            }
            if (self.sessionConnected){
                  self.session!.disconnect()
                  self.session!.close()
            }
}
```

El siguiente método se ejecuta cada vez se sale de la región definida por el beacon y como se puede observar es muy similar a la explicada anteriormente, que se ejecutaba al entrar en la zona definida por el beacon. La principal diferencia es que se realiza la llamada a "EncenderApagar" con el valor necesario para apagar la luz.

```
func beaconManager(manager: AnyObject, didExitRegion region: CLBeaconRegion) {
            print("--> Begin didExitRegion")
            EncenderApagar(false)
            rangingTask =
UIApplication.sharedApplication().beginBackgroundTaskWithExpirationHandler({() -> Void
in})

            myTimer = NSTimer.scheduledTimerWithTimeInterval(10, target: self, selector:
#selector(AppDelegate.myTimerEvent), userInfo: nil, repeats: false)

            print("--> End didExitRegion")
}
```

Como se comentó anteriormente, tanto cuando se entra como cuando se sale de la región se crea la tarea en segundo plano y se inicia un timer de 10 s, que asegura el envío del mensaje por MQTT. Cuando finaliza el timer se ejecuta la segunda función, que básicamente termina la tarea en segundo plano, de modo que la App deja de estar en ejecución y se podría pasar a un modo de bajo consumo.

```
func myTimerEvent(){
            print("My Timer Event")
            print("--> Begin endBackgroundTask")
            UIApplication.sharedApplication().endBackgroundTask(rangingTask)
            print("--> End endBackgroundTask")
}
```

Finalmente, también ha sido necesario definir una función para manejar la sesión MQTT. Dicha función tiene la siguiente codificación.

```
  func handleEvent(session: MQTTSession!, event eventCode: MQTTSessionEvent, error:
NSError!) {
            switch eventCode {
                  case .Connected:
                         sessionConnected = true
                  case .ConnectionClosed:
                         sessionConnected = false
                  default:
                         sessionError = true
                  }
            }
}
```

Como se ha comentado anteriormente, el fichero "ViewController.swift" es la clase controladora de la vista descrita anteriormente con los 4 botones y la etiqueta. Este código fuente es muy similar en algunos puntos al presentado y descrito en el fichero "AppDelegate.swift". En este caso se define la clase "ViewController", que hereda de la clase "UIViewController" e implementa un protocolo asociado a la comunicación MQTT a través de la clase delegada "MQTTSessionDelegate". A continuación se han definido las mismas variables que en el fichero explicado anteriormente y 3 variables de tipo IBOutlet, que sirven para tener acceso desde el código fuente a los controles insertados en la vista.

```
import UIKit
import MQTTClient
import Foundation

class ViewController: UIViewController, MQTTSessionDelegate {

    var sessionConnected = false
    var sessionError = false
    var sessionReceived = false
    var sessionSubAcked = false
    var session : MQTTSession?

    @IBOutlet weak var btnConectar: UIButton!
    @IBOutlet weak var lblEstado: UILabel!
    @IBOutlet weak var btnDesconectar: UIButton!
```

El método "viewDidLoad" se ejecuta cuando la vista ha terminado de cargarse y el usuario ya puede hacer uso de la misma. En este caso, se deshabilita el botón de desconectar y se inicializa la comunicación MQTT con el bróker.

```
    override func viewDidLoad() {
        super.viewDidLoad()

        //let mqttSwift = MQTTSwift()
        //mqttSwift.testSwiftSubscribe()

        btnDesconectar.enabled = false

        session = MQTTSession();
        session!.delegate = self;

        session!.connectToHost("192.168.1.42",
                               port:1883,
                               usingSSL: false);
        while !sessionConnected && !sessionError {
            NSRunLoop.currentRunLoop().runUntilDate(NSDate(timeIntervalSinceNow: 1))
        }

        // Do any additional setup after loading the view, typically from a nib.
    }
```

El método "Apagar" es de tipo IBAction, es decir, se ha creado automáticamente desde la interfaz hacia el código fuente usando la función de pulsar la tecla control y arrastrar (al igual que los elementos de tipo IBOutlet descritos anteriormente). Este método especial se ejecuta cuando el usuario pulsa el botón de apagar colocado en la vista de la App y se publica el valor necesario por MQTT para ejecutar la acción de apagar la luz.

```
    @IBAction func Apagar(sender: AnyObject) {
        if (sessionConnected){
            session!.publishData("1".dataUsingEncoding(NSUTF8StringEncoding,
allowLossyConversion: false),
                                 onTopic: "zona1/izquierda/",
                                 retain: false,
                                 qos: .AtMostOnce)
        }
    }
```

La acción de encender es similar a la anterior y está asociada al botón correspondiente de la vista de la App. De igual manera, también tenemos las funciones "Conectar" y "Desconectar", que están asociadas con los dos botones restantes de la App. Estos métodos también son de tipo IBAction y se han creado automática en el código desde la vista de la App.

```
    @IBAction func Encender(sender: AnyObject) {
        if (sessionConnected){
            session!.publishData("0".dataUsingEncoding(NSUTF8StringEncoding,
allowLossyConversion: false),
                                 onTopic: "zona1/izquierda/",
                                 retain: false,
                                 qos: .AtMostOnce)
        }
    }

    @IBAction func Conectar(sender: AnyObject) {
        session = MQTTSession();
        session!.delegate = self;

        session!.connectToHost("192.168.1.42",
                               port:1883,
                               usingSSL: false);
        while !sessionConnected && !sessionError {
            NSRunLoop.currentRunLoop().runUntilDate(NSDate(timeIntervalSinceNow: 1))
        }
    }

    @IBAction func Desconectar(sender: AnyObject) {
        if (sessionConnected){
            session!.disconnect()
            session!.close()
        }
    }
```

Los tres métodos siguientes responden a la implementacón del protocolo mencionado anteriormente. En dichos métodos se maneja el estado de la sesión, la recepción de un nuevo mensaje y la recepción del denominada subAck.

```
    func handleEvent(session: MQTTSession!, event eventCode: MQTTSessionEvent, error:
NSError!) {
        switch eventCode {
        case .Connected:
            sessionConnected = true
            lblEstado.text = "Conectado..."
            btnDesconectar.enabled = true
            btnConectar.enabled = false
        case .ConnectionClosed:
            sessionConnected = false
            lblEstado.text = "Conexión cerrada..."
            btnDesconectar.enabled = false
            btnConectar.enabled = true
        default:
            sessionError = true
            lblEstado.text = "Error..."
        }
    }

    func newMessage(session: MQTTSession!, data: NSData!, onTopic topic: String!, qos:
MQTTQosLevel, retained: Bool, mid: UInt32) {
        print("Received \(data) on:\(topic) q\(qos) r\(retained) m\(mid)")
        sessionReceived = true;
    }

    func subAckReceived(session: MQTTSession!, msgID: UInt16, grantedQoss qoss:
[NSNumber]!) {
        sessionSubAcked = true;
    }
}
```

Finalmente, el fichero "PuenteObjectiveC.h" es necesario, ya que el SDK de Estimote está escrito en lenguaje ObjectiveC y para integrar ambos lenguajes es necesarios seguir este mecanismo.

```
//
//  PuenteObjectiveC.h
//  MQTTSwift
//
//  Created by Juan Antonio on 7/9/16.
//  Copyright © 2016 OwnTracks. All rights reserved.
//
#ifndef PuenteObjectiveC_h
#define PuenteObjectiveC_h
#import <EstimoteSDK/EstimoteSDK.h>
#endif /* PuenteObjectiveC_h */
```

4.5 Dispositivo 3. Broker.

El dispositivo 3 es el elemento encargado de gestionar la red y transmitir los mensajes. En este caso está basado en un dispositivo empotrado Raspberry Pi 2 modelo B.

4.5.1 Hardware.

El modelo de Raspberry Pi utilizado usa un procesador: Quad-Core Cortex A7 a 900 MHz, tiene una memoria RAM de 1 GB y dispone de los siguientes puertos:

4 x USB 2.0.
1 x 40 GPIO pin.
1 X HDMI.
1 x Ethernet.
1 x Combo audio/mic.
1 x Interfaz de cámara (CSI).
1 X Interfaz de Pantalla (DSI).
1 x Micro SD.
1 x Núcleo Gráfico 3D.

4.5.2 Vista del dispositivo real.

Ilustración 4.11. Raspberry 2 Modelo B

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Ilustración 4.12. Detalle de los pines del conector de la Raspberry 2 Modelo B

Ilustración 4.13. Carcasa de Raspberry 2 Modelo B

4.5.3 Software.

4.5.3.1 Instalación del SO Debian en la Raspberry Pi:

El proceso de instalación del SO Debian en la tarjeta es el siguiente:

1. Descargar el software Berryboot en el PC y descomprimir el fichero.
2. Formatear la tarjeta microSD en formato FAT32 o FAT, para lo cual se puede usar SDFormater, que asegura la compatibilidad con la tarjeta.
3. Copiar los ficheros de Berryboot en el directorio raíz de la tarjeta microSD.
4. Introducir la tarjeta microSD en la tarjeta, conectar teclado, ratón, cable HDMI y la fuente de alimentación.
5. Conectar una unidad USB de 16 GB con la tarjeta.
6. La Raspberry Pi se encenderá automáticamente y aparecerá el menú de configuración de Berryboot, se seleccionan las preferencias fijadas y se pulsa OK.

Las pantallas que irán apareciendo indican el proceso a seguir:

Ilustración 4.14. Instalación de Berryboot

Seleccionamos la unidad donde va a instalarse el sistema operativo de la RBPi (unidad USB, que mejora el rendimiento y la velocidad de ejecución del SO en la RBPi):

Ilustración 4.15. Selección de destino de la instalación

Cargamos el sistema operativo Debian y configuramos los parámetros IP de la red:

Ilustración 4.16. Instalación del SO Debian

Una vez instalado, reiniciamos el sistema y elegimos el SO.

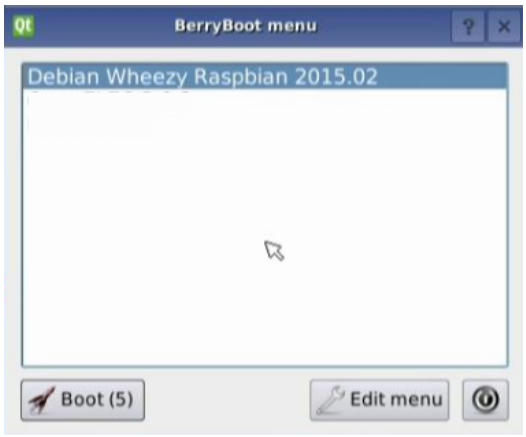


Ilustración 4.17. Elección del SO Debian

Se iniciará el sistema operativo comentado y se podrá utilizar como un microordenador desde el teclado, ratón y monitor HDMI conectados.

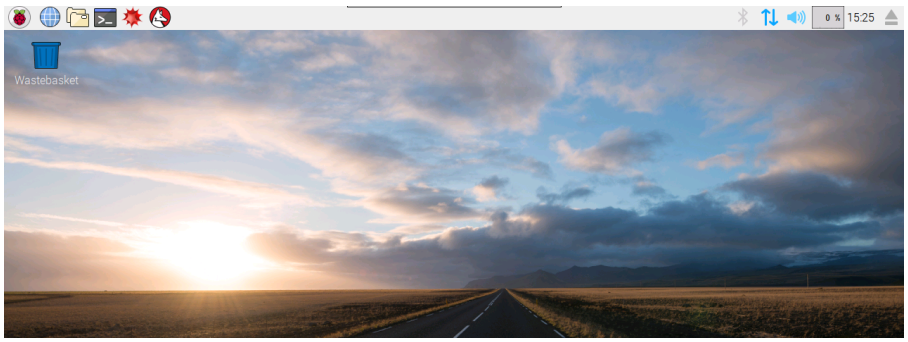


Ilustración 4.18. Interfaz de la RPi con el usuario

4.5.3.2 Acesso a RBPi desde PC: SSH y VNC

Por SSH

Para acceder a la tarjeta mediante SSH se puede utilizar la aplicación PuTTy, con la configuración indicada en la Ilustración 4.19. Básicamente, hay que configurar la IP de la tarjeta, así como el puerto 22, que se corresponde con el protocolo SSH.

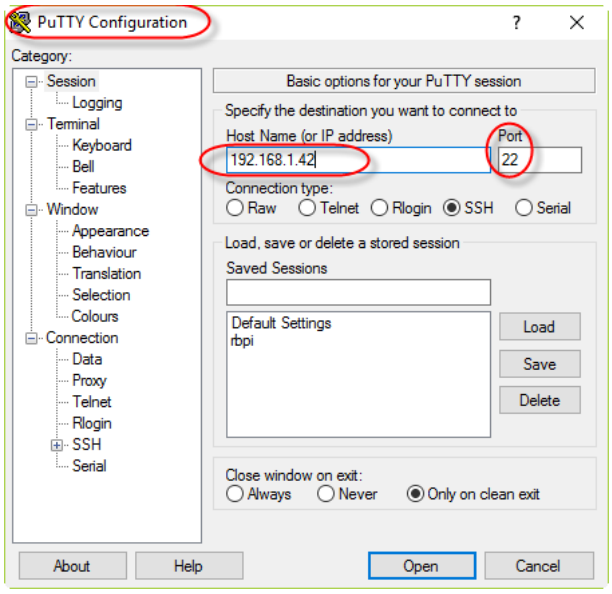


Ilustración 4.19. Configuración de PuTTy

Como resultado, se obtendrá acceso a la Raspberry Pi mediante terminal, una vez introducidos el login y password indicados en la siguiente figura.

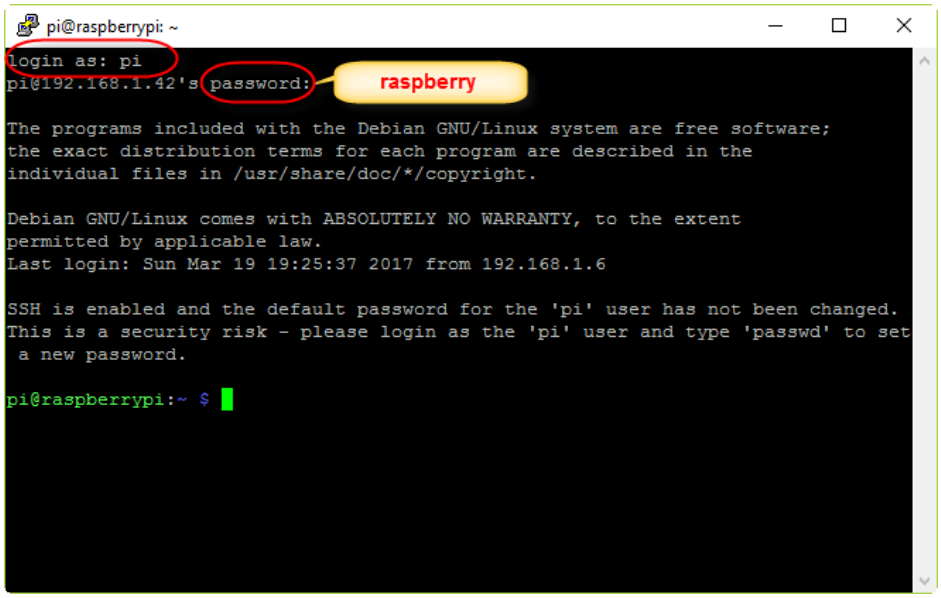


Ilustración 4.20. Accediendo al sistema

Por VNC

El primer paso es descargar e instalar el cliente para el PC. Para el caso del sistema operativo Windows se puede instalar el siguiente: https://www.realvnc.com/download/vnc/windows/

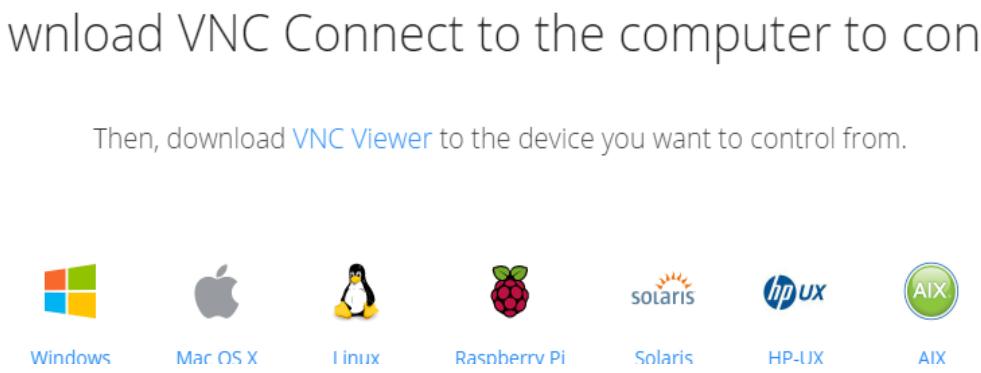


Ilustración 4.21. Página de descarga de VNC

El siguiente paso es instalar el servidor en la Raspberry Pi. En este caso, se ha instalado por SSH ejecutando los siguientes comandos:

- Instalar el servidor de VNC en la tarjeta:

```
sudo apt-get install tightvncserver
```

- Iniciar el servicio VNC en la RBPi:

```
vncserver :1 -geometry 1280x800 -depth 16 -pixelformat rgb565
```

- Acceder desde el PC con la aplicación VNC viewer:

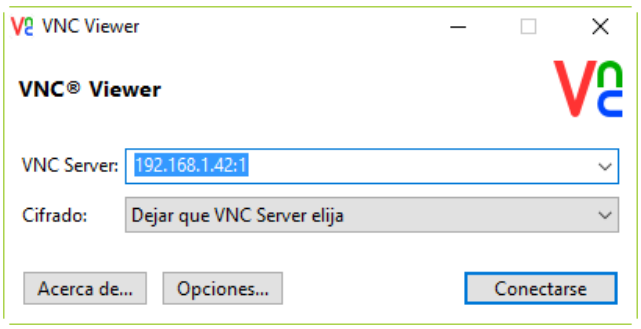


Ilustración 4.22. IP de la raspberry

Ilustración 4.23. Contraseña de acceso

Obteniendo acceso a la tarjeta y pudiendo configurar desde aquí todo el sistema.

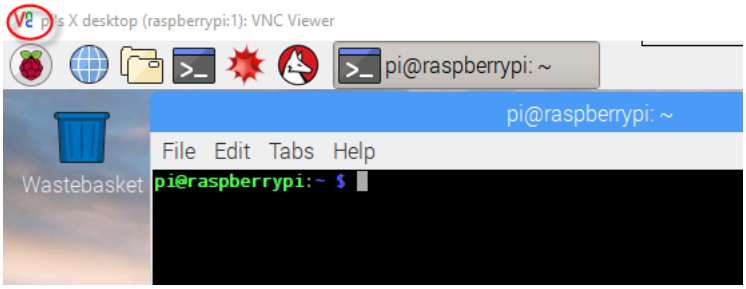


Ilustración 4.24. Acceso al sistema concluido

4.5.3.3 Instalación y configuración del servidor MQTT "Mosquitto"

Desde una terminal abierta en la RBPi, tanto si se realiza por SSH como si se realiza por VNC, la instalación del servidor Mosquito se realiza con las siguientes instrucciones:

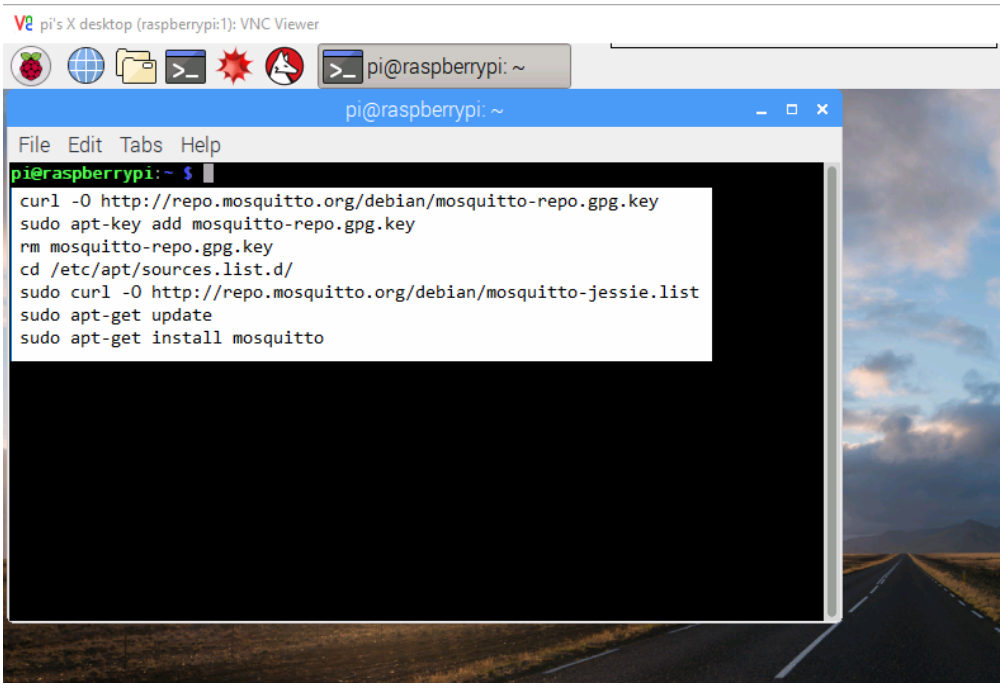


Ilustración 4.25. Instrucciones para instalar el servidor Mosquitto

Una vez instalado Mosquitto en la RBPi, podemos empezar a comunicarnos con los siguientes comandos (abrir 2 consolas, una para suscribir topic y otra para publicar).

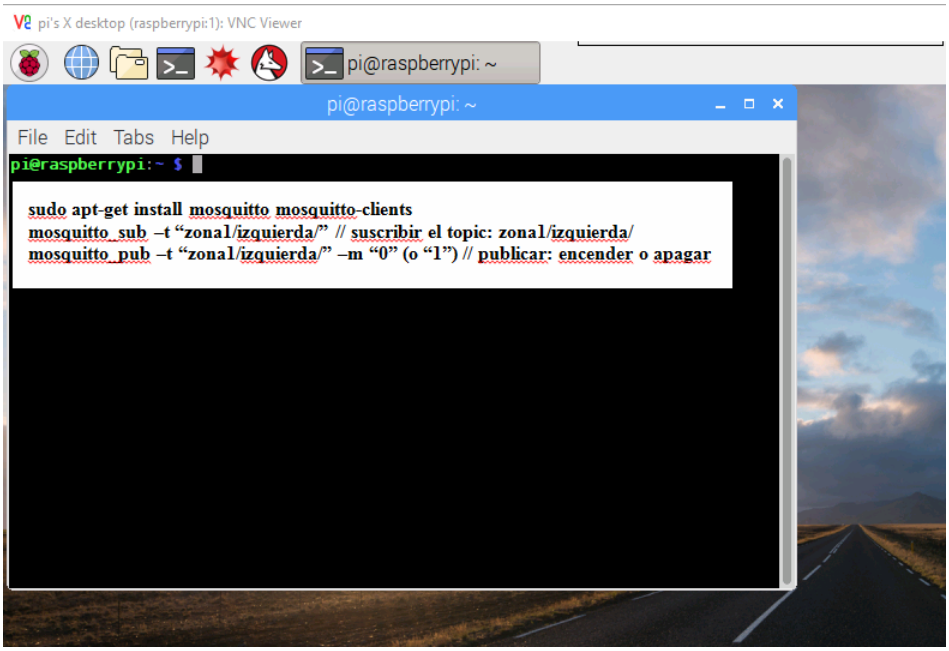


Ilustración 4.26. Comandos de comunicación del Mosquitto

4.5.3.4 Pruebas desde la aplicación de escritorio MQTT.fx

Para probar el buen funcionamiento del servidor Mosquitto, instalamos en nuestro PC la aplicación MQTT.fx, que se utilizará para simular un cliente MQTT.

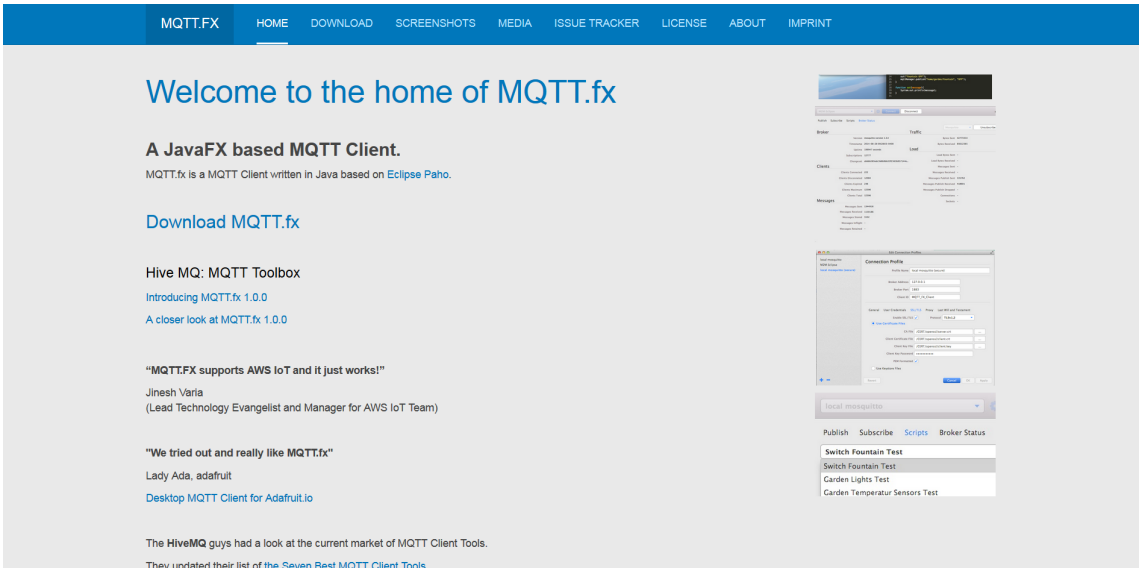


Ilustración 4.27. Descarga de MQTT.fx

Una vez instalada, ya se puede abrir la aplicación anterior y configurar el acceso a Mosquito a la misma. Para ello, habrá que asignar el NOMBRE, la DIRECCIÓN IP (de la RBPi) Y el PUERTO (por defecto, Mosquitto responde en el puerto 1883).

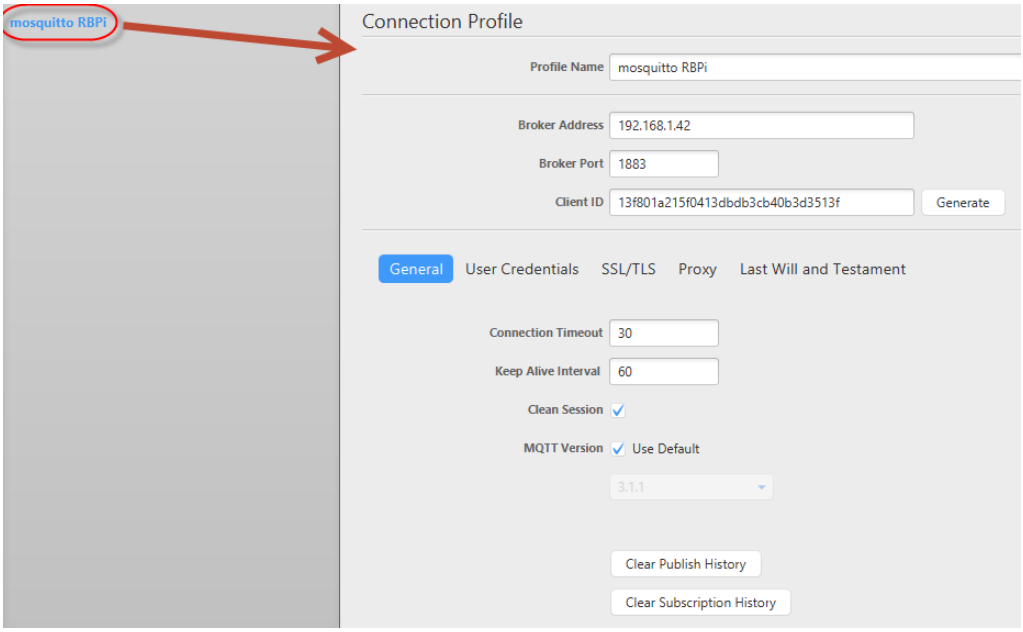


Ilustración 4.28. Configuración de MQTT.fx

Una vez realizada la configuración, se puede pulsar sobre el botón "Connect" para establecer la conexión con el servidor, así como seleccionar el tab "Subscribe" para publicar un mensaje en el topic deseado.

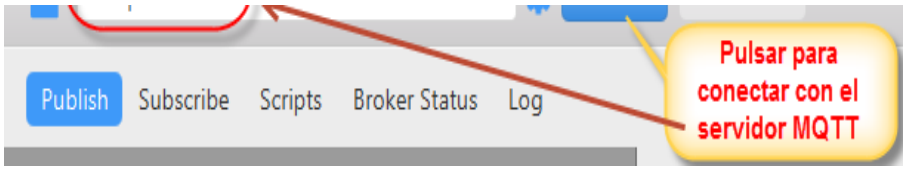


Ilustración 4.29. Conexión con el servidor Mosquito

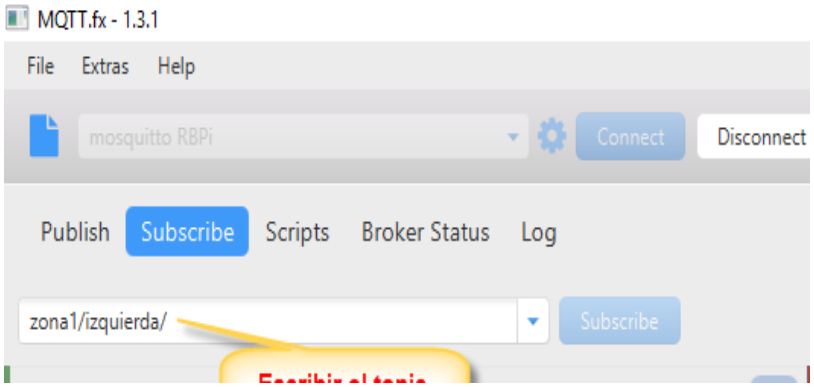


Ilustración 4.30. Suscripción del topic "zona1/izquierda/"

En las dos figuras siguientes se puede ver como mayor detalle los pasos necesarios para publicar un mensaje, así como una ventaja de Log para observar la recepción de los mismos.

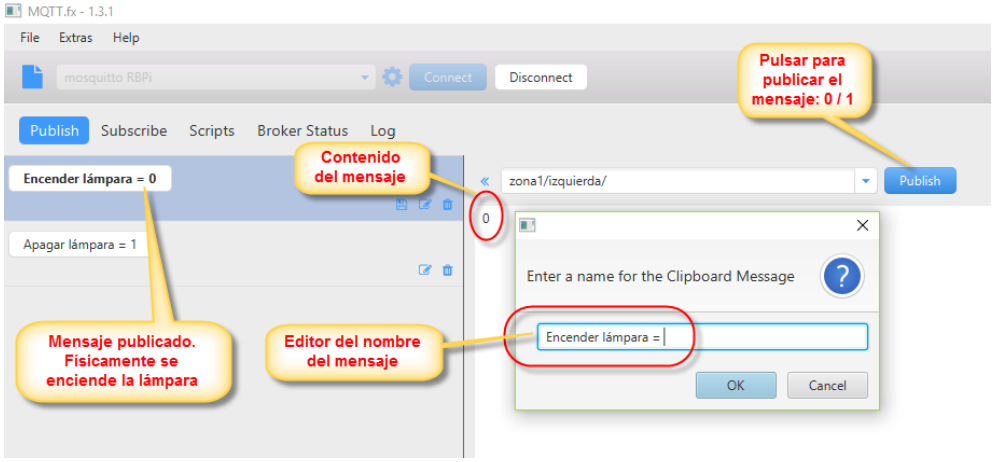


Ilustración 4.31. Publicación del mensaje

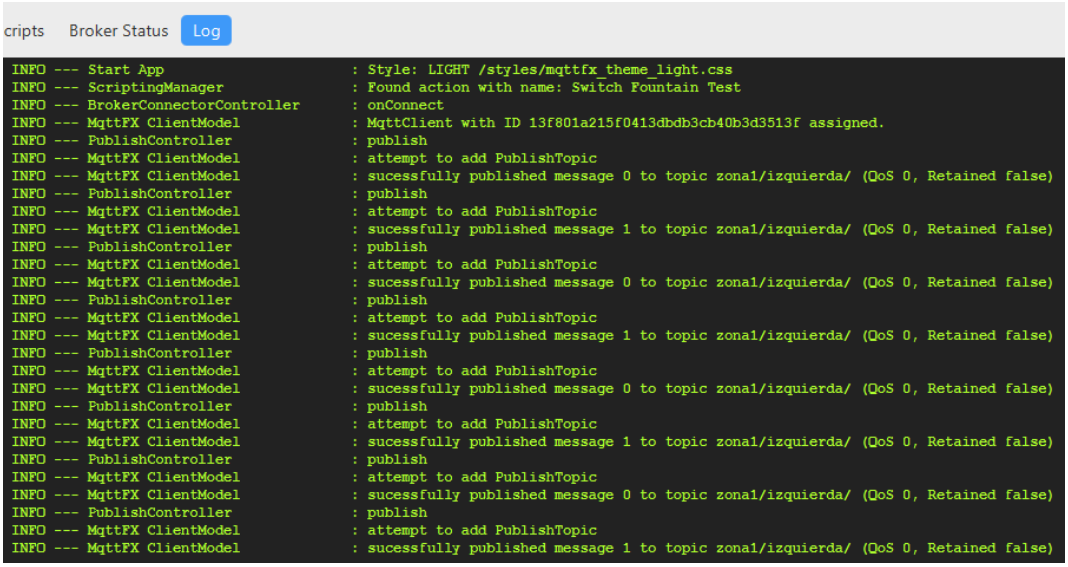


Ilustración 4.32. LOG de publicaciones en Mosquitto (apagar-1 y encender-0 luz)

4.6 Dispositivo 4. Punto de Acceso.

El dispositivo 4 es un punto de acceso, que crea una red inalámbrica que permite interconectar la Raspberry Pi (bróker del sistema) con el dispositivo móvil inteligente que interactúa con los beacons. De este modo, cuando el Smartphone detecta un beacon, realiza el procesamiento necesario y envía la información necesaria al bróker. En este caso se ha utilizado un AP CPE210 de Exterior de 9 dBi de ganancia, que trabaja en la banda de 2,4 GHz y permite transferencias de hasta 300 Mbps.

4.6.1 Hardware.

El modulo incorpora antenas MIMO doblemente polarizadas y direccionales de 9 dBi (de ganancia) 2x2. Su potencia de transmisión es ajustable desde 0 hasta 27 dBm/500mw. La tecnología TP-LINK Pharos MAXtream TDMA (Acceso Múltiple por División en el Tiempo) se encarga del rendimiento del producto, en capacidad y latencia, siendo ideal para aplicaciones PTMP.

El dispositivo mencionado se puede configurar con los siguientes modos de operación: AP / Cliente / Bridge/ Repetidor / Router AP / Router Cliente AP Client (WISP).

4.6.2 Vista real del dispositivo.

Ilustración 4.33. Módulo AP CPE210

4.7 Dispositivo 5. Modulo WiFi.

Como método de comunicación entre la luminaria y el bróker se ha usado un módulo WiFi embebido, concretamente el módulo ESP8266. Además, con este tipo de módulo existen tarjetas de acople para aumentar número de pines. En nuestro ha utilizado la tarjeta NodeMCU.

4.7.1 Hardware.

NodeMcu es una iniciativa open-source basada en el ESP8266, que es un montaje que se puede colocar en una protoboard y además, incluye un conector mini USB para

programar el chip interno y comunicarse con el PC si es necesario, con lo que se evita el la necesidad de usar un adaptador FTDI-USB, que emula un puerto serie en el PC.

NodeMcu soporta, tanto el modo Arduino como un modo propio de desarrollo con un lenguaje reminiscente de Basic, que pueden ser cargados mediante actualizaciones firmware.

En la siguiente figura se resumen las características principales de la tarjeta NodeMCU:

- Voltaje de entrada (USB): 5V
- Voltaje de salida en los pines: 3.3V
- Voltaje de referencia en el ADC: 3.3V
- Corriente nominal por pin: 12mA
- Frecuencia de procesador: 80MHz (160MHz max.)
- 4MB Flash
- Consumo de corriente en stand-by @80MHz: 80mA
- Consumo de corriente al recibir una peticion (librería WebServer en modo de punto
  80MHz: 90mA
- Consumo de corriente al utilizar HTTPClient.get() @ 80 MHz: 100-110mA
- Consumo de corriente en stand-by @160MHz: 90mA
- Consumo de corriente al recibir una peticion (librería WebServer en modo de punto

Ilustración 4.34. Características de la tarjeta NodeMCU

4.7.2 Vista real del dispositivo.

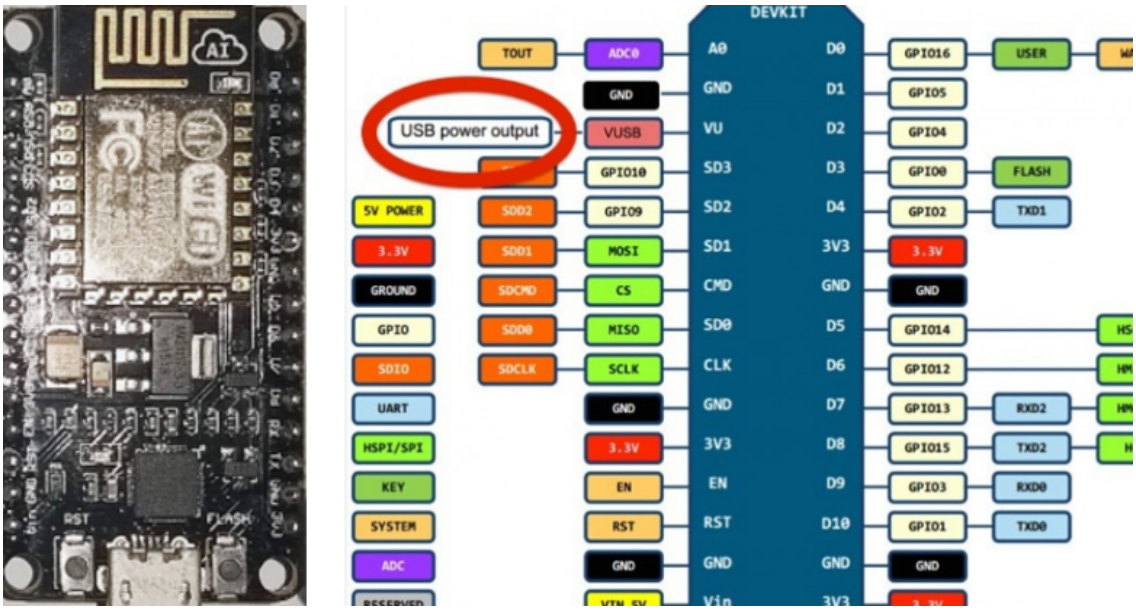


Ilustración 4.35. Foto y esquema de la tarjeta

4.7.3 Software.

Una vez instalada la tarjeta en el entorno de Arduino, se podrá seleccionar la misma accediendo al menú "Herramientas" y a la opción "Placa". Dentro de este menú se podrá seleccionar la tarjeta "NodeMCU 1.0 (ESP-12E Module)".

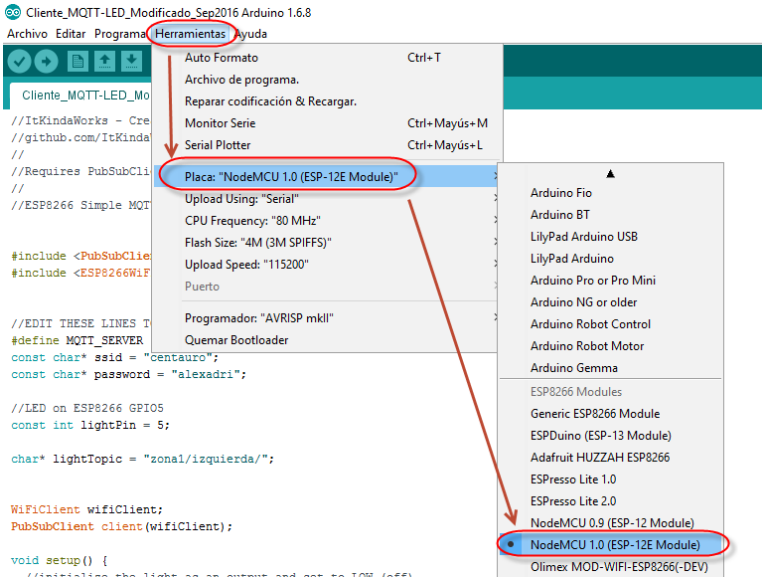


Ilustración 4.36. Selección de la tarjeta en el IDE de Arduino

A continuación, se va a describir el código que se ha desarrollado para el módulo ESP8266, que como se ha comentado anteriormente es el encargado de activar y desactivar la bombilla que tenga conectada a través de un relé.

La siguiente figura muestra que se ha definido la IP del bróker, así como los datos necesarios para establecer la conexión inalámbrica con el punto de acceso. También se ha definido el pin con el que está conectado el relé que controla la bombilla, así como la definición del topic asociado y descrito anteriormente.

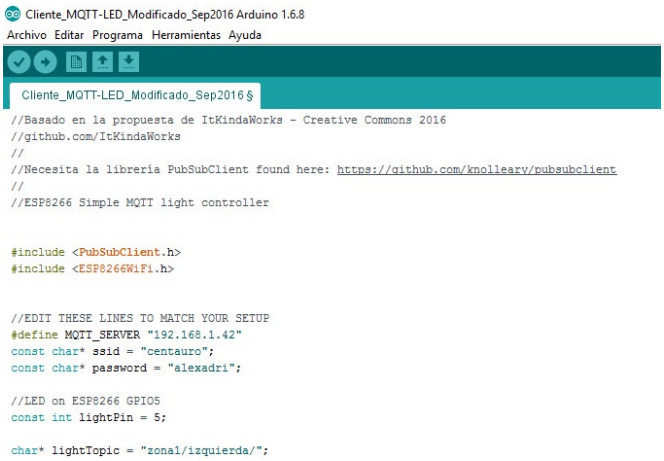


Ilustración 4.37. Cabecera y definición de constantes y la variable del topic MQTT

En la siguiente ilustración se muestra el código asociado a la declaración de un objeto de la clase "WiFiClient", así como al objeto necesario para intercambiar tramas por MQTT y la definición del método "setup()". En concreto, dentro de este método se configura el pin para que inicialmente este apagado, luego comienza la depuración. A continuación, se inicia la conexión con el servidor por el puerto 1883, para lo cual se utilizan los valores de SSID y clave definidos anteriormente.

```
WiFiClient wifiClient;
PubSubClient client(wifiClient);

void setup() {
  //initialize the light as an output and set to LOW (off)
  pinMode(lightPin, OUTPUT);
  digitalWrite(lightPin, LOW);

  //start the serial line for debugging
  Serial.begin(115200);
  delay(100);

  //star MQTT server
  client.setServer (MQTT_SERVER, 1883);
  client.setCallback(callback);

  //start wifi subsystem
  WiFi.begin(ssid, password);
  //attempt to connect to the WIFI network and then connect to the MQTT server
  reconnect();

  //wait a bit before starting the main loop
      delay(2000);
```

Ilustración 4.38. Configuración del cliente wifi y el servidor MQTT

Dentro del método loop se comienza un bucle de conexión permanente (client.loop). Además, si se recibe la publicación de un mensaje correcto a través de MQTT (método callback), se enciende o se apaga la luz, según el valor publicado en el topic.

```
void loop(){

  //reconnect if connection is lost
  if (!client.connected() && WiFi.status() == 3) {reconnect();}

  //maintain MQTT connection
  client.loop();

  //MUST delay to allow ESP8266 WIFI functions to run
  delay(10);
}

void callback(char* topic, byte* payload, unsigned int length) {

  //convert topic to string to make it easier to work with
  String topicStr = topic;

  //Print out some debugging info
  Serial.println("Callback update.");
  Serial.print("Topic: ");
  Serial.println(topicStr);

  //turn the light on if the payload is '1' and publish to the MQTT server a confirmation message
  if(payload[0] == '1'){
    digitalWrite(lightPin, HIGH);
    client.publish("/zona1/izquierda/", "Light On");
  }

  //turn the light off if the payload is '0' and publish to the MQTT server a confirmation message
  else if (payload[0] == '0'){
    digitalWrite(lightPin, LOW);
    client.publish("/test/confirm", "Light Off");
  }
```

Ilustración 4.39. Lazo principal

Para el correcto funcionamiento del proceso es importante que la conexión a la red WiFi no se pierda y se realice una reconexión automática en caso de que dicha conexión se haya perdido.

```
void reconnect() {

  //attempt to connect to the wifi if connection is lost
  if(WiFi.status() != WL_CONNECTED){
    //debug printing
    Serial.print("Connecting to ");
    Serial.println(ssid);

    //loop while we wait for connection
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }

    //print out some more debug once connected
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
  }
```

Ilustración 4.40. Reconexión con la red WiFi

El siguiente listado muestra que cuando se establece la conexión con la red WiFi se comprueba si el cliente no está conectado con el servidor MQTT. En caso afirmativo, se genera el nombre del cliente a través de la dirección MAC del dispositivo y la conexión al topic deseado. Finalmente, se intenta la conexión y la subscripción con el topic.

```
//make sure we are connected to WIFI before attemping to reconnect to MQTT
if(WiFi.status() == WL_CONNECTED){
// Loop until we're reconnected to the MQTT server
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");

    // Generate client name based on MAC address and last 8 bits of microsecond counter
    String clientName;
    clientName += "esp8266-";
    uint8_t mac[6];
    WiFi.macAddress(mac);
    clientName += macToStr(mac);

    //if connected, subscribe to the topic(s) we want to be notified about
    if (client.connect((char*) clientName.c_str())) {
      Serial.print("\tMTQQ Connected - ");
      client.subscribe(lightTopic);
    }

    //otherwise print failed for debugging
    else{Serial.println("\tFailed."); abort();}
  }
 }
}
```

Ilustración 4.41. Proceso de conexión con el servidor MQTT

Finalmente, se muestra la función utilizada para generar el nombre a la hora de establecer la conexión con el servidor MQTT.

```
//generate unique name from MAC addr
String macToStr(const uint8_t* mac){

  String result;

  for (int i = 0; i < 6; ++i) {
    result += String(mac[i], 16);

    if (i < 5){
      result += ':';
    }
  }

  return result;
}
```

Ilustración 4.42. Generación del nombre del cliente en base a la dirección MAC

Capítulo 5

Conclusiones y Trabajos futuros.

5.1 Conclusiones

En este proyecto se ha abordado el estudio de sistemas automatizados. Concretamente, se ha diseñado un sistema de iluminación inteligente, basado en dispositivos empotrados de bajo coste, eficiente, y que podría resultar de gran utilidad, tanto para el usuario como para el gerente de la empresa. Además de un ahorro de energía beneficioso para todos. Dicho sistema de iluminación ha sido el caso de estudio.

Se ha realizado un estudio para escoger los dispositivos empotrados y la programación de estos más adecuada. Para los dispositivos empotrados, se ha tenido en cuenta el coste, su facilidad de integración, sus prestaciones y su grado de extensión en el panorama actual. Concretamente, se ha seleccionado Raspberry Pi para desarrollar el bróker MQTT. En el caso de la comunicación entre dispositivos, se ha elegido los módulos ESP8266 por su simplicidad y efectividad. Para la detección del usuario se ha escogido la tecnología *Beacon* por su interés en la aplicación en otros campos.

Se ha llevado a cabo un caso de estudio en el que se ha diseñado y ejecutado un modelo de iluminación inteligente. Para ello, se ha implementado el hardware necesario mediante la integración de sensores a los dispositivos empotrados y se ha obtenido el software mediante el desarrollo de los archivos pertinentes. En el caso del módulo ESP8266, se ha creado el Sketch necesario para implementar todos los componentes software necesarios. Finalmente, se ha creado una aplicación en Swift que aporta la app para

Smartphone. Con ella se pueden llevar a cabo la detección del usuario y la comunicación a través de protocolo MQTT con el sistema. Se ha utilizado este protocolo de comunicación por su uso de ancho de banda mínimo, por su poco consumo de energía, por su rapidez, que posibilita un tiempo de respuesta superior al resto de protocolos web actuales y su gran fiabilidad y poco requisito de recursos y memorias

Finalmente, y a modo de resumen, con este estudio se ha pretendido comprobar la utilidad de uno de los muchos servicios posibles que pueden ser creados mediante la tecnología de sistemas inteligentes. La creciente evolución de las ciudades hacia la inteligencia y la automatización, da muestra del verdadero potencial de ésta tecnología.

5.2 Trabajos Futuros

Como posible trabajo futuro, se puede crear una aplicación de mayor dimensión mediante la implementación de más *beacons* sensores que aumenten la zona afectada. Por otro lado, se pueden llevar a cabo estudios más exhaustivos de optimización para crear sistemas de control y registro de la propia iluminación.

El caso de estudio llevado a cabo es completamente funcional y es posible implementarlo en el cualquier espacio de trabajo y ocio. Sin embargo, como se ha citado anteriormente, esta es una pequeña muestra del potencial de esta tecnología y su capacidad de diagnosticar un sistema. La luminaria es solo un ejemplo de aquello que puede automatizarse en un sistema inteligente.

Finalmente, otro aspecto que podría ser explorado en el futuro es el diseño y desarrollo de la app para el sistema operativo Android, ya que, actualmente sólo se ha realizado para el sistema operativo iOS mediante el lenguaje de programación Swift.

Bibliografía

[Android] Historia del Sistema operativo. Disponible on-line en: https://www.android.com/intl/es_es/history/#/marshmallow

[Arduino] Guía oficial del producto. Disponible on-line en: https://www.arduino.cc/en/Guide/HomePage

[BeagleBone] BeagleBone wiki. Disponible on-line en: http://beagleboard.org/project/Beagle+wiki/

[Bluetooth] What is Bluetooth? Disponible on-line en: https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth

[ESP8266] Tutoriales e información sobre el módulo. Disponible on-line en: http://www.prometec.net/esp8266/

[HTTP] HTTp tutorial. Disponible on-line en: http://www.tutorialspoint.com/http/

[iBeacon] iBeacon for developers. Disponible on-line en: https://developer.apple.com/ibeacon/

[iOS] iOS' history. Disponible on-line en: http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad

[Intel Compute Stick] Descripción y análisis del Stick. Disponible on-line en: http://www.xataka.com/analisis/intel-compute-stick-analisis

[MQTT] Documentation about MQTT. Disponible on-line en: http://mqtt.org/documentation

[NFC] NFC FORUM. Disponible on-line en: http://nfc-forum.org/

[RFID] What is RFID? Disponible on-line en: http://www.technovelgy.com/ct/technology-article.asp

[RPi] RPi educative Forum. Disponible on-line en: https://www.raspberrypi.org/forums/

[WiFi] Artículo sobre esta tecnología. Disponible on-line en: http://www.aulaclic.es/articulos/wifi.html

[Windows Phone] A history of Windows Pone. Disponible on-line en: https://mspoweruser.com/a-history-of-windows-phone-the-road-to-threshold/